```cpp
#include <iostream>  // C++ I/O
#include <fstream>  // File I/O
#include <sstream>  // String stream I/O
#include <iomanip>  // C++ I/O manipulator

#include <cstdlib>  // C library
#include <cstdio>  // C I/O
#include <ctime>  // C time
#include <cmath>  // Math library
#include <cstring>  // C strings

#include <vector>  // Vector
#include <queue>  // Queue
#include <stack>  // Stack
#include <map>  // Map
#include <set>  // Set
#include <algorithm>  // Algorithms

using namespace std;

#define reps(_var,_begin,_end,_step)  for (int _var = (_begin);  \
    _var <= (_end); _var += (_step))
#define reps_(_var,_end,_begin,_step)  for (int _var = (_end);  \
    _var >= (_begin); _var -= (_step))
#define rep(_var,_begin,_end)  reps(_var, _begin, _end, 1)
#define rep_(_var,_end,_begin)  reps_(_var, _end, _begin, 1)
#define minimize(_var,_targ)  _var = min(_var, _targ)
#define maximize(_var,_targ)  _var = max(_var, _targ)

typedef unsigned long long ull;
typedef long long lli, ll;
typedef double llf;

template <typename typ>
void memclr(typ p) {
    memset(p, 0, sizeof(p)); }
template <typename typ>
void memclr(typ arr[], int n) {
    memset(arr, 0, sizeof(arr[0]) * (n + 1)); }
template <typename typ, int dim>
void memclr(typ arr[][dim], int n, int m) {
    rep(i, 0, n) memset(arr[i], 0, sizeof(arr[i][0]) * (m + 1)); }

lli read(void)
{
    lli res = 0, sgn = 1;
    char ch = getchar();
    while(ch < '0' || ch > '9')
        sgn = ch == '-' ? -1 : 1, ch = getchar();
    while(ch >= '0' && ch <= '9')
        res = res * 10 + ch - '0', ch = getchar();
    return res * sgn;
}

const int maxn = 1010;

int main(int argc, char** argv)
{
    return 0;
}


// fm_begin(maths, euclid_gcd)
// fm_begin(maths, chinese_remainder_theorem)
// fm_begin(maths, fast_exponentiation)
// fm_begin(maths, prime_filter)
// fm_begin(maths, fermats_little_theorem)
// fm_begin(maths, miller_rabin)
// fm_begin(strings, trie)
// fm_begin(strings, knuth_morris_pratt)
// fm_begin(strings, aho_corasick_automaton)
// fm_begin(strings, suffix_array)
// fm_begin(strings, suffix_automaton)
// fm_begin(strings, manacher)
// fm_begin(trees, disjoint_set)
// fm_begin(trees, segment_tree)
// fm_begin(trees, splay)
// fm_begin(trees, kd_tree)
// fm_begin(graphs, basic_graph)
// fm_begin(graphs, dijkstra)
// fm_begin(graphs, floyd_warshall)
// fm_begin(graphs, tree_diameter)
// fm_begin(graphs, tree_center)
// fm_begin(graphs, heavy_light_decomposition)
// fm_begin(graphs, link_cut_tree)
// fm_begin(graphs, prim)
// fm_begin(graphs, kruskal)
// fm_begin(graphs, scc_tarjan)
// fm_begin(graphs, dcc_tarjan_v)
// fm_begin(graphs, dcc_tarjan_e)
// fm_begin(graphs, 2_sat)
// fm_begin(graphs, dinic)
// fm_begin(graphs, spfa_costflow)
// fm_begin(graphs, zkw_costflow)
// fm_begin(graphs, hungary_match)
// fm_begin(graphs, bron_kerbosch)
// @chapter 1.1 Point 定义
// @chapter 1.2 Line 定义
// @chapter 1.3 两点间距离
// @chapter 1.4 判断 线段相交
// @chapter 1.5 判断 直线和线段相交
// @chapter 1.6 点到直线距离
// @chapter 1.7 点到线段距离
// @chapter 1.8 计算多边形面积
// @chapter 1.9 判断点在线段上
// @chapter 1.10 判断点在凸多边形内
// @chapter 1.11 判断点在任意多边形内
// @chapter 1.12 判断多边形
// @chapter 1.13 简单极角排序
// @chapter 2.1 凸包
// @chapter 3.1 平面最近点对
// @chapter 4.1 旋转卡壳 / 平面最远点对
// @chapter 4.2 旋转卡壳计算平面点集最大三角形面积
// @chapter 4.3 求解两凸包最小距离
// @chapter 5.1 半平面交
// @chapter 6.1 三点求圆心坐标
// @chapter 7.1 求两圆相交的面积


fm_begin(maths, euclid_gcd):
    // @desc  Euclidean greatest common divisor algorithm.
    // @complexity  Time: O(Log[n]), Space: O(n)
    // @usage  gcd(a, b): calculate (a, b)
    // @usage  lcm(a, b): calculate [a, b]
    // @usage  extended_gcd(a, b, x, y): solve equation ax + by = gcd(a, b)
    //         and store results in x, y (such x, y always exists)
    lli gcd(lli a, lli b)
    {
        if (b == 0)
            return a;
        return gcd(b, a % b);
    }
    lli lcm(lli a, lli b)
    {
        return a / gcd(a, b) * b;
    }
    lli extended_gcd(lli a, lli b, lli& x, lli& y)
    {
        if (b == 0) {
            x = 1, y = 0;
            return a;
        }
        int q = extended_gcd(b, a % b, y, x);
        y -= lli(a / b) * x;
        return q;
    }
fm_end(maths, euclid_gcd);
```

```
fm_begin(maths, chinese_remainder_theorem):
    // @desc  Chinese remainder theorem
    // @complexity  Time: O(n Log[n]), Space: O(n)
    // @usage  solve(a[], m[], n): solve a series of equation, s.t.
    //         x \equiv a_i (mod m_i)  \forall i = 1..n
    //         m_i has to be coprime with each other
    // @usage  extended_solve(a[], m[], n): solve a series of equation,
s.t.
    //         x \equiv a_i (mod m_i)  \forall i = 1..n
    //         m_i doesn't have to be coprime
    //         returns -1 if no solutions available
    fm(maths, euclid_gcd) egcd;
    lli solve(lli a[], lli m[], int n)
    {
        lli res = 0, lcm = 1, t, tg, x, y;
        rep(i, 1, n)
            lcm *= m[i];
        rep(i, 1, n) {
            t = lcm / m[i];
            egcd.extended_gcd(t, m[i], x, y);
            x = ((x % m[i]) + m[i]) % m[i];
            res = (res + t * x * a[i]) % lcm;
        }
        return (res + lcm) % lcm;
    }
    lli extended_solve(lli a[], lli m[], int n)
    {
        lli cm = m[1], res = a[1], x, y;
        rep(i, 2, n) {
            lli A = cm, B = m[i], C = (a[i] - res % B + B) % B,
                gcd = egcd.extended_gcd(A, B, x, y),
                Bg = B / gcd;
            if (C % gcd != 0)
                return -1;
            x = (x * (C / gcd)) % Bg;
            res += x * cm;
            cm *= Bg;
            res = (res % cm + cm) % cm;
        }
        return (res % cm + cm) % cm;
    }
fm_end(maths, chinese_remainder_theorem);


fm_begin(maths, fast_exponentiation):
    // @desc  Sped up exponential calculation
    // @complexity  Time: O(Log[n]), Space: O(n)
    // @usage  pow(a, k, m): calculate a^k % m
    // @usage  pow(a, k): calculate a^k (potential overflow)
    lli pow(lli a, lli k, lli m = 0)
    {
        lli res = 1, tmp = a;
        while (k > 0) {
            if ((k & 1) == 1) {
                res *= tmp;
                if (m > 0)
                    res %= m;
            }
            k >>= 1;
            tmp *= tmp;
            if (m > 0)
                tmp %= m;
        }
        return res;
    }
fm_end(maths, fast_exponentiation);


fm_begin(maths, prime_filter):
    // @desc  Filter prime numbers
    // @complexity  Time: O(n), Space: O(n)
    // @usage  isprime[i]: true if i is a prime number, elsewise false
    // @usage  primes[0]: number of prime numbers
    // @usage  primes[i]: The i-th prime number
    fm_const(int, maxn, 100000000);
    bool isprime[maxn];
    int primes[maxn];
    void filter(void)
    {
        isprime[1] = false;
        rep(i, 2, maxn - 1)
            isprime[i] = true;
        primes[0] = 0;
        rep(i, 2, maxn - 1) {
            if (!isprime[i])
                continue;
            reps(j, i, maxn - 1, i)
                isprime[j] = false;
            primes[++primes[0]] = i;
        }
        return ;
    }
fm_end(maths, prime_filter);


fm_begin(maths, fermats_little_theorem):
    // @desc  Fermat's little theorem
    //        if IsPrime[p] and Gcd[a, p] == 1:
    //            a ^ (p - 1) === 1  (mod p)
    //            (a ^ (p - 1)) % p = 1
    // @complexity  Time: O(Log[n]), Space: O(1)
    // @usage  calc(a, p, k): calculate a^p % k
    lli calc(lli a, lli p, lli k)
    {
        // Asserted: IsPrime[p] and Gcd[a, p] = 1
        if (p < k - 1)
            return lli(pow(a, p)) % k;
        return calc(a, p % (k - 1), k);
    }
fm_end(maths, fermats_little_theorem);


fm_begin(maths, miller_rabin):
    // @desc  Miller-Rabin prime testing
    //        relies on Fermat's little theorem
    // @complexity  Time: O(k Log[n]^2), Space: O(1)
    // @usage  test(n, k): test n under modulo k
    // @usage  is_prime(n): true if n is prime, elsewise false
    fm(maths, fast_exponentiation) fexp;
    bool test(lli n, lli k)
    {
        if (fexp.pow(k, n - 1, n) != 1)
            return false;
        lli t = n - 1, tmp;
        while (t % 2 == 0) {
            t >>= 1;
            tmp = fexp.pow(k, t, n);
            if (tmp != 1 && tmp != n - 1)
                return false;
            if (tmp == n - 1)
                return true;
        }
        return true;
    }
    bool is_prime(lli n)
    {
        if (n == 1 || (n > 2 && n % 2 == 0))
            return false;
        lli samples[14] = {4,
            2, 3, 5, 7,  // n < 3.2e9
            11, 13, 17, 19, 23, 29, 31, 37,  // n < 1.8e19
            41,  // n < 3.3e25
        };
        rep(i, 1, samples[0]) {
            if (n == samples[i])
                return true;
```

```
        if (n > samples[i] && !test(n, samples[i]))
            return false;
    }
    return true;  // Certain prime
  }
fm_end(maths, miller_rabin);


fm_begin(strings, trie):
    // @desc  Trie tree, string indexer
    // @complexity  Time: O(n), Space: O(Sum[n])
    // @usage  __makenode(): creates empty node
    // @usage  __insert(p, str, level): inserts str into tree, recursively
    //         returns the node marking termination of this string
    // @usage  __remove(p, str, level): removes str from tree, recursively
    //         returns true if such string exists and is removed
    // @usage  __query(p, str, level): returns the node marking termination
of
    //         string str
    // @usage  init(): initializes tree
    // @usage  insert(str, data): inserts str into tree, if the string
already
    //         exists, original data will be replaced by new one instead
    // @usage  remove(str): removes str from tree, returns true if and only
if
    //         the string existed and is removed
    // @usage  query(str, data): query if str existed and its data, returns
    //         true if string existed, and its key is stored in data.
    fm_const(int, max_nodes, 1001000);
    fm_const(int, charset_size, 256);
    struct node
    {
        int flag, children;
        node *child[charset_size];
        void *data;
    };
    node npool[max_nodes], *root;
    int npcnt;
    node* __make_node(void)
    {
        node *p = &npool[++npcnt];
        p->flag = false;
        p->children = 0;
        memclr(p->child);
        p->data = nullptr;
        return p;
    }
    node* __insert(node* p, const string& str, int level)
    {
        if (level == str.length()) {
            if (!p->flag)
                p->children += 1;
            p->flag = true;
            return p;
        }
        char ch = str[level];
        if (!p->child[ch])
            p->child[ch] = __make_node();
        p->children -= p->child[ch]->children;
        node *q = __insert(p->child[ch], str, level + 1);
        p->children += p->child[ch]->children;
        return q;
    }
    bool __remove(node* p, const string& str, int level)
    {
        if (level == str.length()) {
            if (p->flag) {
                p->children -= 1;
                p->flag = false;
                return true;
            }
            return false;
        }
        char ch = str[level];
        if (p->child[ch] == nullptr)
            return false;
        p->children -= p->child[ch]->children;
        bool res = __remove(p->child[ch], str, level + 1);
        p->children += p->child[ch]->children;
        if (p->child[ch]->children == 0)
            p->child[ch] = nullptr;
        return res;
    }
    node* __query(node* p, const string& str, int level)
    {
        if (level == str.length())
            return p;
        char ch = str[level];
        if (!p->child[ch])
            return nullptr;
        return __query(p->child[ch], str, level + 1);
    }
    void init(void)
    {
        npcnt = 0;
        root = __make_node();
        return ;
    }
    bool insert(const string& str, void* data = nullptr)
    {
        node *p = __insert(root, str, 0);
        if (!p)
            return false;
        p->data = data;
        return true;
    }
    bool remove(const string& str)
    {
        return __remove(root, str, 0);
    }
    bool query(const string& str, void*& data)
    {
        node *p = __query(root, str, 0);
        if (p == nullptr || !p->flag)
            return false;
        data = p->data;
        return true;
    }
    bool query(const string& str)
    {
        void *data;
        return query(str, data);
    }
fm_end(strings, trie);


fm_begin(strings, knuth_morris_pratt):
    // @desc  Knuth-Morris-Pratt string matching algorithm
    // @complexity  Time: O(n+m), Space: O(m)
    // @usage  src[], n: base string and its length, indices starts from 0
    // @usage  pat[], m: pattern and length, indices starts from 0
    // @usage  get_next(): retrieve next[] array for pattern
    // @usage  match(begin): match next occurence starting from begin
    fm_const(int, max_len, 100100);
    int n, m, src[max_len], pat[max_len], next[max_len];
    void get_next(void)
    {
        next[0] = -1;
        rep(i, 1, m - 1)
            for (int j = next[i - 1]; ; j = next[j]) {
                if (pat[j + 1] == pat[i]) {
                    next[i] = j + 1;
                    break;
                } else if (j == -1) {
                    next[i] = -1;
                    break;
                }
            }
```

```
        return ;
    }
    int match(int begin = 0)
    {
        int i = begin, j = 0;
        for (; i < n && j < m; ) {
            if (src[i] == pat[j]) {
                i += 1;
                j += 1;
            } else if (j == 0) {
                i += 1;
            } else {
                j = next[j - 1] + 1;
            }
        }
        if (j == m)
            return i - m;
        return -1;
    }
fm_end(strings, knuth_morris_pratt);


fm_begin(strings, aho_corasick_automaton):
    // @desc  Aho-Corasick automaton, match a series of patterns in a
string
    // @complexity  Time: O(n+Sum[m]), Space: O(Sum[m])
    // @usage  match_res: match result: [(position in string, pattern id)]
    // @usage  __make_node(): create new empty node
    // @usage  __insert(p, str, str_id, level): insert string, recursively
    // @usage  init(): initialize empty tree
    // @usage  insert(str, str_id): insert str into tree, marking its id as
    //         str_id. Only the first of same strings would appear in the
tree
    // @usage  build_tree(): construct fail pointers, no further inserts
should
    //         appear after build_tree, and no matches shall precede this
    // @usage  match(str): find all occurences of strings in tree in string
    //         and store the result in a match_res object
    fm_const(int, max_nodes, 1001000);
    fm_const(int, charset_size, 256);
    typedef vector<pair<int, int>> match_res;
    struct node
    {
        int val, flag, flag_len, children;
        node *child[charset_size], *parent, *fail;
        node *first_child, *next;  // Adjacency list
    };
    node npool[max_nodes], *root;
    int npcnt;
    node* __make_node(void)
    {
        node *p = &npool[++npcnt];
        p->val = p->flag = p->flag_len = p->children = 0;
        memclr(p->child);
        p->parent = p->fail = nullptr;
        p->first_child = p->next = nullptr;
        return p;
    }
    node* __insert(node* p, const string& str, int str_id, int level)
    {
        if (level == str.length()) {
            if (p->flag > 0)
                return nullptr;
            p->flag = str_id;
            p->flag_len = str.length();
            return p;
        }
        char ch = str[level];
        if (p->child[ch] == nullptr) {
            node *q = p->child[ch] = __make_node();
            q->val = ch;
            q->parent = p;
            q->next = p->first_child;
            p->first_child = q;
        }
        return __insert(p->child[ch], str, str_id, level + 1);
    }
    void init(void)
    {
        npcnt = 0;
        root = __make_node();
        root->fail = root;
        return ;
    }
    void insert(const string& str, int str_id)
    {
        __insert(root, str, str_id, 0);
        return ;
    }
    void build_tree(void)
    {
        queue<node*> que;
        root->fail = root;
        for (node *np = root->first_child; np; np = np->next) {
            np->fail = root;
            for (node *mp = np->first_child; mp; mp = mp->next)
                que.push(mp);
        }
        while (!que.empty()) {
            node *p = que.front();
            que.pop();
            p->fail = p->parent->fail->child[p->val];
            if (p->fail == nullptr)
                p->fail = root;
            for (node *np = p->first_child; np; np = np->next)
                que.push(np);
        }
        return ;
    }
    match_res match(const string& str)
    {
        match_res res;
        int pos = 0;
        node *p = root;
        while (pos <= str.length()) {
            char ch = str[pos];
            if (p->flag > 0)
                res.push_back(make_pair(pos - p->flag_len, p->flag));
            if (pos == str.length())
                break;
            while (p->child[ch] == nullptr && p != root)
                p = p->fail;
            if (p->child[ch] != nullptr)
                p = p->child[ch];
            pos += 1;
        }
        return res;
    }
fm_end(strings, aho_corasick_automaton);


fm_begin(strings, suffix_array):
    // @desc  Suffix array
    // @warning  incompatible code style
    // 喜欢钻研问题的 JS 同学，最近又迷上了对加密方法的思考. 一天，他突
然想出了
    // 一种他认为是终极的加密办法: 把需要加密的信息排成一圈，显然，它
们有很多种
    // 不同的读法.
    // JSOI07 SOI07J OI07JS I07JSO 07JSOI 7JSOI0 把它们按照字符串的大小排
序:
    // 07JSOI 7JSOI0 I07JSO JSOI07OI07JS SOI07J 读出最后一列字符: I0O7SJ,
    // 就是加密后的字符串 (其实这个加密手段实在很容易破解，鉴于这是突
然想出来
    // 的，那就^^). 但是，如果想加密的字符串实在太长,...
    fm_const(int, maxn, 800100);
    // This suffix array is only used to be sorted.
    class SuffixArray
```

```
{
public:
    int n, pwn, rank[maxn], prank[maxn];
    int stra[maxn], strb[maxn], srta[maxn], srtb[maxn];
    int posa[maxn], posb[maxn], cnt[maxn];
    void init(int _n, int* arr)
    {
        n = _n;
        pwn = 1; while (pwn < n) pwn <<= 1;
        for (int i = 1; i <= n; i++)
            rank[i] = arr[i];
        for (int i = n + 1; i <= pwn; i++)
            rank[i] = 0;
        swap(n, pwn);
        return ;
    }
    void build(void)
    {
        int c = max(n, 257);
        for (int d = 1; d <= n; d <<= 1) {
            // Initialize "string" to be compared...
            rep (i, 1, n)   stra[i] = rank[i],
                            strb[i] = rank[i + d];
            // Resetting counter for bit-2 to be sorted
            memclr(cnt);
            rep (i, 1, n)   cnt[strb[i]]++;
            rep (i, 1, c)   cnt[i] += cnt[i - 1];
            // To sort according to second position
            rep (i, 1, n)   srta[cnt[strb[i]]] = stra[i],
                            srtb[cnt[strb[i]]] = strb[i],
                            posb[cnt[strb[i]]--] = i;
            // Resetting counter for bit-1 to be sorted
            memclr(cnt);
            rep (i, 1, n)   cnt[srta[i]]++;
            rep (i, 1, c)   cnt[i] += cnt[i - 1];
            rep_ (i, n, 1)  stra[cnt[srta[i]]] = srta[i],
                            strb[cnt[srta[i]]] = srtb[i],
                            posa[cnt[srta[i]]--] = posb[i];
            // Re-updating rank array and continue
            rep (i, 1, n)   rank[posa[i]] = stra[i] == stra[i - 1] &&
                                strb[i] == strb[i - 1]
                    ? rank[posa[i - 1]] : rank[posa[i - 1]] + 1;
            continue;
        }
        swap(n, pwn);
        // Reverse rank[] to be assigned / positioned easily.
        for (int i = 1; i <= n; i++)
            prank[rank[i]] = i;
        return ;
    }
} sa;
int     n, arr[maxn];
char    str[maxn], res[maxn];
void download(void)
{
    // To retrieve the sorting procedures and send to output pipeline.
    int pos = 0;
    for (int i = 1; i <= 2 * n; i++) {
        if (sa.prank[i] > n)
            continue;
        // Assign result position's value
        res[++pos] = arr[sa.prank[i] + n - 1];
    }
    return ;
}
void main(void)
{
    scanf("%s", str);
    n = strlen(str);
    rep(i, 1, n) arr[i + n] = arr[i] = (int)str[i - 1];
    // Done copying... now suffix sorting.
    sa.init(n * 2, arr);
    sa.build();
    // Downloading final results and output...
```

```
        download();
        for (int i = 1; i <= n; i++)
            printf("%c", (char)res[i]);
        return ;
    }
fm_end(strings, suffix_array);


fm_begin(strings, suffix_automaton):
    // @desc  Suffix automaton
    // @warning  incompatible code style
    // @warning  not yet understood
    fm_const(int, MAXN, 100100);
    struct NODE
    {
        int ch[26];
        int len,fa;
        NODE(){memset(ch,0,sizeof(ch));len=0;}
    }dian[MAXN<<1];
    int las=1,tot=1;
    void add(int c)
    {
        int p=las;int np=las=++tot;
        dian[np].len=dian[p].len+1;
        for(;p&&!dian[p].ch[c];p=dian[p].fa)dian[p].ch[c]=np;
        if(!p)dian[np].fa=1;//以上为case 1
        else
        {
            int q=dian[p].ch[c];
            if(dian[q].len==dian[p].len+1)dian[np].fa=q;//以上为case 2
            else
            {
                int nq=++tot;dian[nq]=dian[q];
                dian[nq].len=dian[p].len+1;
                dian[q].fa=dian[np].fa=nq;
                for(;p&&dian[p].ch[c]==q;p=dian[p].fa)dian[p].ch[c]=nq;
                //以上为case 3
            }
        }
    }
    char s[MAXN];int len;
    void main()
    {
        scanf("%s",s);len=strlen(s);
        for(int i=0;i<len;i++)add(s[i]-'a');
    }
fm_end(strings, suffix_automaton);


fm_begin(strings, manacher):
    // @desc  Manacher algorithm, calculates longest palindrome in string
    // @complexity  Time: O(n), Space: O(n)
    // @usage  eval(s, begin, length): returns the longest palindrome in s
and
    //          sets begin and length as result
    fm_const(int, maxn, 100100);
    int n, str[maxn], p[maxn];
    string eval(string s, int& begin, int& length)
    {
        n = s.length();
        str[0] = -1;
        rep(i, 1, n) {
            str[2 * i - 1] = s[i - 1];
            str[2 * i] = -1;  // Some unused char
        }
        int mx = 0, id = 0, res_len = 0, res_center = 0;
        rep(i, 1, 2 * n) {
            p[i] = mx > i ? min(p[2 * id - 1], mx - i) : 1;
            while (str[i + p[i]] == str[i - p[i]])
                p[i] += 1;
            if (mx < i + p[i]) {
                mx = i + p[i];
                id = i;
            }
```

```
            if (res_len < p[i]) {
                res_len = p[i];
                res_center = i;
            }
        }
        begin = (res_center - res_len) / 2;
        length = res_len;
        return s.substr(begin, length - 1);
    }
fm_end(strings, manacher);


fm_begin(trees, disjoint_set):
    // @desc  Disjoint set
    // @complexity  Time: O(1), Space: O(n)
    // @usage  init(n, w[]): set n objects with weight w[1..n]
    // @usage  find(p): find the component id (parent of this component)
    // @usage  join(p, q): join component p with q
    // @usage  val[find(p)]: find weight of component p
    // @usage
    fm_const(int, maxn, 1001000);
    int n, par[maxn], size[maxn];
    lli val[maxn];
    void init(int n, lli w[] = nullptr)
    {
        rep(i, 1, n) {
            par[i] = i;
            size[i] = 1;
        }
        if (w != nullptr)
            rep(i, 1, n)
                val[i] = w[i];
        return ;
    }
    int find(int p)
    {
        if (par[p] != p) {
            int q = find(par[p]);
            par[p] = q;
        }
        return par[p];
    }
    void join(int p, int q)
    {
        int gp = find(p), gq = find(q);
        if (size[gq] < size[gp])
            swap(gq, gp);
        par[gp] = gq;
        val[gq] += val[gp];  // @modify
        size[gq] += size[gp];
        return ;
    }
fm_end(trees, disjoint_set);


fm_begin(trees, segment_tree):
    // @desc  Segment tree
    // @complexity  Time: O(n Log[n]), Space: O(n)
    // @warning  incompatible code style
    // @warning  disorganized functions
    // 给出n个数，有两个操作，第一个操作是将区间[x,y]中的数都开根号，
    // 第二个操作是求区间[x,y]的和。
    fm_const(int, maxn, 200100);
    struct node
    {
        node *lc, *rc;
        int lb, mb, rb;
        lli sum;
    } *root, npool[maxn<<1];
    int n, ncnt;
    node* make_node(void)
    {
        node *p = &npool[++ncnt];
        p->lc = p->rc = NULL;
        p->lb = p->mb = p->rb = 0;
        p->sum = 0;
        return p;
    }
    lli query(node *p, int l, int r)
    {
        if (p->lb == l && p->rb == r) {
            return p->sum;
        }
        if (r <= p->mb) {
            return query(p->lc, l, r);
        } else if (l > p->mb) {
            return query(p->rc, l, r);
        } else {
            return query(p->lc, l, p->mb) +
                query(p->rc, p->mb + 1, r);
        }
        return lli();
    }
    lli query(int l, int r)
    {
        if (l > r) swap(l, r);
        return this->query(root, l, r);
    }
    void change(node *p, int l, int r)
    {
        if (p->lb == l && p->rb == r) {
            if (p->rb - p->lb + 1 == p->sum)
                return ;
            if (p->lb == p->rb) {
                p->sum = sqrt(p->sum);
                return ;
            }
            change(p->lc, l, p->mb);
            change(p->rc, p->mb + 1, r);
            p->sum = p->lc->sum + p->rc->sum;
            return ;
        }
        if (r <= p->mb) {
            change(p->lc, l, r);
        } else if (l > p->mb) {
            change(p->rc, l, r);
        } else {
            change(p->lc, l, p->mb);
            change(p->rc, p->mb + 1, r);
        }
        p->sum = p->lc->sum + p->rc->sum;
        return ;
    }
    void change(int l, int r)
    {
        if (l > r) swap(l, r);
        this->change(root, l, r);
        return ;
    }
    node* build_tree(int l, int r, lli arr[])
    {
        node *p = make_node();
        int mid = (l + r) >> 1;
        p->lb = l; p->mb = mid; p->rb = r;
        if (p->lb == p->rb) {
            p->sum = lli(arr[mid]);
        } else {
            p->lc = build_tree(l, mid, arr);
            p->rc = build_tree(mid + 1, r, arr);
            p->sum = p->lc->sum + p->rc->sum;
        }
        return p;
    }
    void build(int n, lli arr[])
    {
        this->ncnt = 0;
        this->n = n;
        this->root = this->build_tree(1, n, arr);
```

```
        return ;
    }
fm_end(trees, segment_tree);


fm_begin(trees, splay):
    // @desc  Splay tree (poj3580)
    // @complexity  Time: O(n Log[n]), Space: O(n)
    // @usage  main(): test function
    // @warning  this method is still incomplete, problems may exist,
please
    //          check the robustness before using this
    // @warning  code style
    fm_const(int, maxn, 10010);
    fm_const(int, infinit, 1000000007);
    int ch[maxn][2], parent[maxn], root, ncnt, n;
    int size[maxn], val[maxn], sum[maxn], minn[maxn];
    int lazyadd[maxn], lazyswp[maxn];
    #define lc(x) ch[x][lazyswp[x]]
    #define rc(x) ch[x][!lazyswp[x]]
    #define par(x) parent[x]
    int makenode(int q, int v)
    {
        int p = ++ncnt; n++;
        lc(p) = rc(p) = 0;
        par(p) = q;
        size[p] = 1;
        val[p] = sum[p] = minn[p] = v;
        lazyadd[p] = lazyswp[p] = 0; // Initially they aren't lazy at all
        return p;
    }
    void updateminn(int p)
    {
        minn[p] = p > 2 ? val[p] : infinit;
        if (lc(p)) minn[p] = min(minn[p], minn[lc(p)]);
        if (rc(p)) minn[p] = min(minn[p], minn[rc(p)]);
        return ;
    }
    void dispatchlazyadd(int p)
    {
        // Separate dispatched lazy values to children
        lazyadd[lc(p)] += lazyadd[p];
        lazyadd[rc(p)] += lazyadd[p];
        // Update children's initial values
        val[lc(p)] += lazyadd[p];
        val[rc(p)] += lazyadd[p];
        // Update children's sums
        sum[lc(p)] += size[lc(p)] * lazyadd[p];
        sum[rc(p)] += size[rc(p)] * lazyadd[p];
        // Update minimum queried values
        minn[lc(p)] += lazyadd[p];
        minn[rc(p)] += lazyadd[p];
        // Finally reset lazy value
        lazyadd[p] = 0;
        return ;
    }
    bool dispatchlazyswp(int p)
    {
        if (!lazyswp[p]) return false;
        lazyswp[lc(p)] ^= 1;
        lazyswp[rc(p)] ^= 1;
        swap(lc(p), rc(p));
        lazyswp[p] = 0;
        return true;
    }
    void rotate(int p)
    {
        int q = par(p), g = par(q), x = p == rc(q);
        // Dispatching lazy values in case something goes wrong
        dispatchlazyadd(q);
        dispatchlazyadd(p);
        if (dispatchlazyswp(q)) x ^= 1; // These make no modifications to
the
                                        // actual values
        dispatchlazyswp(p);
        // Relink connexions between nodes
        ch[q][x] = ch[p][!x], par(ch[q][x]) = q;
        ch[p][!x] = q, par(q) = p;
        par(p) = g;
        if (g) ch[g][rc(g) == q] = p;
        // Update data values
        size[q] = size[lc(q)] + 1 + size[rc(q)];
        size[p] = size[lc(p)] + 1 + size[rc(p)];
        sum[q] = sum[lc(q)] + val[q] + sum[rc(q)];
        sum[p] = sum[lc(p)] + val[p] + sum[rc(p)];
        updateminn(p);
        updateminn(q);
        return ;
    }
    void splay(int p, int t)
    {
        for (int q = 0; (q = par(p)) && q != t; rotate(p))
            if (par(q) && par(q) != t)
                rotate((p == lc(q)) == (q == lc(par(q))) ? q : p);
        if (t == 0) root = p;
        return ;
    }
    int suc(int p)
    {
        if (!rc(p)) { while (p == rc(par(p))) p = par(p); p = par(p); }
        else { p = rc(p); while (lc(p)) p = lc(p); }
        return p;
    }
    int find(int x)
    {
        int p = root;
        while (true) {
            if (x <= size[lc(p)]) {
                p = lc(p);
                continue;
            } x -= size[lc(p)];
            if (x <= 1)
                return p;
            x -= 1;
            p = rc(p);
        }
        return 0;
    }
    void insert(int x, int v)
    {
        int lp = find(x), rp = suc(lp); // Operations should be guranteed
that
                                        // rp is valid
        splay(rp, 0);
        splay(lp, root);
        int c = makenode(lp, v);
        rc(lp) = c;
        size[lp]++, sum[lp] += v;
        size[rp]++, sum[rp] += v;
        updateminn(lp);
        updateminn(rp);
        return ;
    }
    void remove(int x)
    {
        int lp = find(x - 1), rp = suc(x);
        splay(rp, 0);
        splay(lp, root);
        int c = rc(lp);
        size[lp]--, sum[lp] -= val[c];
        size[rp]--, sum[rp] -= val[c];
        updateminn(lp);
        updateminn(rp);
        n--;
        return ;
    }
    int query_sum(int l, int r)
    {
```

```
    int lp = find(l - 1), rp = find(r + 1);
    splay(rp, 0);
    splay(lp, root);
    // Return data values
    return sum[rc(lp)];
}
int query_min(int l, int r)
{
    int lp = find(l - 1), rp = find(r + 1);
    splay(rp, 0);
    splay(lp, root);
    // Return data values
    return minn[rc(lp)];
}
void modify_add(int l, int r, int v)
{
    int lp = find(l - 1), rp = find(r + 1);
    splay(rp, 0);
    splay(lp, root);
    // Update data values
    sum[rc(lp)] += size[rc(lp)] * v;
    sum[lp] += size[rc(lp)] * v;
    sum[rp] += size[rc(lp)] * v;
    minn[rc(lp)] += v;
    val[rc(lp)] += v;
    printf("$ modify_add: it is %d who's talking about\n", rc(lp));
    updateminn(lp);
    updateminn(rp);
    lazyadd[rc(lp)] += v;
    return ;
}
void modify_swp(int l, int r)
{
    int lp = find(l - 1), rp = find(r + 1);
    splay(rp, 0);
    splay(lp, root);
    // Updating data values, which were easier
    lazyswp[rc(lp)] ^= 1;
    return ;
}
void buildtree()
{
    n = ncnt = 0;
    root = makenode(0, 0);
    rc(root) = makenode(root, 0);
    minn[1] = minn[2] = infinit;
    par(rc(root)) = root;
    size[root]++;
    return ;
}
#undef lc
#undef rc
#undef par
void test_main(void)
{
    buildtree();
    printf("Program begun.\n");
    while (true)
    {
        string a;
        int b, c, d;
        cin >> a;
        if (a == "insert") {
            cin >> b >> c;
            insert(b + 1, c);
        } else if (a == "delete") {
            cin >> b;
            remove(b + 1);
        } else if (a == "sum") {
            cin >> b >> c;
            printf("sum %d %d = %d\n", b, c, query_sum(b + 1, c + 1));
        } else if (a == "min") {
            cin >> b >> c;
            printf("min %d %d = %d\n", b, c, query_min(b + 1, c + 1));
        } else if (a == "add") {
            cin >> b >> c >> d;
            modify_add(b + 1, c + 1, d);
        } else if (a == "reverse") {
            cin >> b >> c;
            modify_swp(b + 1, c + 1);
        } else if (a == "revolve") {
            cin >> b >> c;
            d = query_sum(c + 1, c + 1);
            insert(b, d);
        }
    }
    return ;
}
fm_end(trees, splay);


fm_begin(trees, kd_tree):
    // @desc  K-D tree
    // @warning  incompatible code style
    // @warning  yet not understood
    /*function of this program: build a 2d tree using the input training
data
    the input is exm_set which contains a list of tuples (x,y)
    the output is a 2d tree pointer*/
    struct data
    {
        double x = 0;
        double y = 0;
    };
    struct Tnode
    {
        struct data dom_elt;
        int split;
        struct Tnode * left;
        struct Tnode * right;
    };
    bool cmp1(data a, data b){
        return a.x < b.x;
    }
    bool cmp2(data a, data b){
        return a.y < b.y;
    }
    bool equal(data a, data b){
        if (a.x == b.x && a.y == b.y)
        {
            return true;
        }
        else{
            return false;
        }
    }
    void ChooseSplit(data exm_set[], int size, int &split, data
&SplitChoice){
        /*compute the variance on every dimension. Set split as the
dismension
        that have the biggest
        variance. Then choose the instance which is the median on this
split
        dimension.*/
        /*compute variance on the x,y dimension. DX=EX^2-(EX)^2*/
        double tmp1,tmp2;
        tmp1 = tmp2 = 0;
        for (int i = 0; i < size; ++i)
        {
            tmp1 += 1.0 / (double)size * exm_set[i].x * exm_set[i].x;
            tmp2 += 1.0 / (double)size * exm_set[i].x;
        }
        double v1 = tmp1 - tmp2 * tmp2;  //compute variance on the x
dimension
        tmp1 = tmp2 = 0;
        for (int i = 0; i < size; ++i)
        {
            tmp1 += 1.0 / (double)size * exm_set[i].y * exm_set[i].y;
```

```
        tmp2 += 1.0 / (double)size * exm_set[i].y;
    }
    double v2 = tmp1 - tmp2 * tmp2;  //compute variance on the y
dimension
    split = v1 > v2 ? 0:1; //set the split dimension
    if (split == 0)
    {
        sort(exm_set,exm_set + size, cmp1);
    }
    else{
        sort(exm_set,exm_set + size, cmp2);
    }
    //set the split point value
    SplitChoice.x = exm_set[size / 2].x;
    SplitChoice.y = exm_set[size / 2].y;
}
Tnode* build_kdtree(data exm_set[], int size, Tnode* T){
    //call function ChooseSplit to choose the split dimension and split
pnt
    if (size == 0){
        return NULL;
    }
    else{
        int split;
        data dom_elt;
        ChooseSplit(exm_set, size, split, dom_elt);
        data exm_set_right [100];
        data exm_set_left [100];
        int sizeleft ,sizeright;
        sizeleft = sizeright = 0;
        if (split == 0)
        {
            for (int i = 0; i < size; ++i)
            {
                if (!equal(exm_set[i],dom_elt) &&
                        exm_set[i].x <= dom_elt.x)
                {
                    exm_set_left[sizeleft].x = exm_set[i].x;
                    exm_set_left[sizeleft].y = exm_set[i].y;
                    sizeleft++;
                }
                else if (!equal(exm_set[i],dom_elt) &&
                        exm_set[i].x > dom_elt.x)
                {
                    exm_set_right[sizeright].x = exm_set[i].x;
                    exm_set_right[sizeright].y = exm_set[i].y;
                    sizeright++;
                }
            }
        }
        else{
            for (int i = 0; i < size; ++i)
            {
                if (!equal(exm_set[i],dom_elt) &&
                        exm_set[i].y <= dom_elt.y)
                {
                    exm_set_left[sizeleft].x = exm_set[i].x;
                    exm_set_left[sizeleft].y = exm_set[i].y;
                    sizeleft++;
                }
                else if (!equal(exm_set[i],dom_elt) &&
                        exm_set[i].y > dom_elt.y)
                {
                    exm_set_right[sizeright].x = exm_set[i].x;
                    exm_set_right[sizeright].y = exm_set[i].y;
                    sizeright++;
                }
            }
        }
        T = new Tnode;
        T->dom_elt.x = dom_elt.x;
        T->dom_elt.y = dom_elt.y;
        T->split = split;
        T->left = build_kdtree(exm_set_left, sizeleft, T->left);
```

```
        T->right = build_kdtree(exm_set_right, sizeright, T->right);
        return T;
    }
}
double Distance(data a, data b){
    double tmp = (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
    return sqrt(tmp);
}
void searchNearest(Tnode * Kd, data target, data &nearestpoint,
        double & distance){
    //1. 如果Kd是空的，则设dist为无穷大返回
    //2. 向下搜索直到叶子结点
    stack<Tnode*> search_path;
    Tnode* pSearch = Kd;
    data nearest;
    double dist;
    while(pSearch != NULL)
    {
        //pSearch加入到search_path中;
        search_path.push(pSearch);
        if (pSearch->split == 0)
        {
            if(target.x <= pSearch->dom_elt.x) /* 如果小于就进入左子树
*/
            {
                pSearch = pSearch->left;
            }
            else
            {
                pSearch = pSearch->right;
            }
        }
        else{
            if(target.y <= pSearch->dom_elt.y) /* 如果小于就进入左子树
*/
            {
                pSearch = pSearch->left;
            }
            else
            {
                pSearch = pSearch->right;
            }
        }
    }
    //取出search_path最后一个赋给nearest
    nearest.x = search_path.top()->dom_elt.x;
    nearest.y = search_path.top()->dom_elt.y;
    search_path.pop();
    dist = Distance(nearest, target);
    //3. 回溯搜索路径
    Tnode* pBack;
    while(search_path.size() != 0)
    {
        //取出search_path最后一个结点赋给pBack
        pBack = search_path.top();
        search_path.pop();
        if(pBack->left == NULL && pBack->right == NULL)
        { /* 如果pBack为叶子结点 */
            if( Distance(nearest, target) >
                    Distance(pBack->dom_elt, target) )
            {
                nearest = pBack->dom_elt;
                dist = Distance(pBack->dom_elt, target);
            }
        }
        else
        {
            int s = pBack->split;
            if (s == 0)
            {
                if( fabs(pBack->dom_elt.x - target.x) < dist)
                {  /* 如果以target为中心的圆（球或超球），半径为dist
的圆与
                            分割超平面相交， 那么就要跳到另一边的子空间去
```

```
搜索 */
                    if( Distance(nearest, target) >
                        Distance(pBack->dom_elt, target) )
                    {
                        nearest = pBack->dom_elt;
                        dist = Distance(pBack->dom_elt, target);
                    }
                    if(target.x <= pBack->dom_elt.x) /* 如果target位于
                        pBack的左子空间，那么就要跳到右子空间去
搜索 */
                        pSearch = pBack->right;
                    else
                        pSearch = pBack->left; /* 如果target位于pBack的
右
                        子空间，那么就要跳到左子空间去搜索 */
                    if(pSearch != NULL)
                        //pSearch加入到search_path中
                        search_path.push(pSearch);
                }
            }
            else {
                if( fabs(pBack->dom_elt.y - target.y) < dist) /* 如果以
                    target为中心的圆（球或超球），半径为dist的圆
与分割
                    超平面相交， 那么就要跳到另一边的子空间去搜
索 */
                {
                    if( Distance(nearest, target) >
                        Distance(pBack->dom_elt, target) )
                    {
                        nearest = pBack->dom_elt;
                        dist = Distance(pBack->dom_elt, target);
                    }
                    if(target.y <= pBack->dom_elt.y) /* 如果target位于
                        pBack的左子空间，那么就要跳到右子空间去
搜索 */
                        pSearch = pBack->right;
                    else
                        pSearch = pBack->left; /* 如果target位于pBack的
                        右子空间，那么就要跳到左子空间去搜索 */
                    if(pSearch != NULL)
                    // pSearch加入到search_path中
                        search_path.push(pSearch);
                }
            }
        }
    }
    nearestpoint.x = nearest.x;
    nearestpoint.y = nearest.y;
    distance = dist;
}
void main(){
    data exm_set[100]; //assume the max training set size is 100
    double x,y;
    int id = 0;
    cout<<"Please input the training data in the form x y." <<
        " One instance per line. Enter -1 -1 to stop."<<endl;
    while (cin>>x>>y){
        if (x == -1)
        {
            break;
        }
        else{
            exm_set[id].x = x;
            exm_set[id].y = y;
            id++;
        }
    }
    struct Tnode * root = NULL;
    root = build_kdtree(exm_set, id, root);
    data nearestpoint;
    double distance;
    data target;
    cout <<"Enter search point"<<endl;
    while (cin>>target.x>>target.y)
    {
        searchNearest(root, target, nearestpoint, distance);
        cout<<"The nearest distance is "<<distance<<
            ",and the nearest point is "<<nearestpoint.x<<","<<
            nearestpoint.y<<endl;
        cout <<"Enter search point"<<endl;
    }
}
fm_end(trees, kd_tree);


fm_begin(graphs, basic_graph):
    // @desc  Basic graph
    // @complexity  Time: O(m), Space: O(m)
    // @usage  init(): clear graph
    // @usage  add_edge(u, v, len): creates directed edge
    // @usage  add_edge_bi(u, v, len): creates undirected edge
    fm_const(int, maxn, 1010);
    fm_const(int, maxm, 1001000);
    struct edge
    {
        int u, v;
        lli len;
        edge *next, *rev;
    };
    edge epool[maxm], *edges[maxn];
    int ecnt;
    edge* add_edge(int u, int v, lli len)
    {
        edge *p = &epool[++ecnt];
        p->u = u; p->v = v; p->len = len;
        p->next = edges[u]; edges[u] = p;
        p->rev = nullptr;
        return p;
    }
    void add_edge_bi(int u, int v, lli len)
    {
        edge *p = add_edge(u, v, len),
            *q = add_edge(v, u, len);
        p->rev = q; q->rev = p;
        return ;
    }
    void init(void)
    {
        ecnt = 0;
        memclr(edges);
        return ;
    }
fm_end(graphs, basic_graph);


fm_begin(graphs, dijkstra):
    // @desc  Shortest Path: Dijkstra
    //        suitable for single-source multi-target positive-weight
graphs
    // @complexity  Time: O(m Log[n]), Space: O(m)
    // @usage  dist[i]: the distance from source to node i
    // @usage  add_edge(u, v, len): create directed edge
    // @usage  eval(s): calculate all distances from source s
    fm_const(int, maxn, 100100);
    fm_const(int, maxm, 1001000);
    fm_const(lli, infinit, 0x007f7f7f7f7f7f7fll);
    struct edge
    {
        int u, v;
        lli len;
        edge *next;
    };
    edge epool[maxm], *edges[maxn];
    int n, ecnt;
    lli dist[maxn];
    typedef pair<lli, int> pli;
    void add_edge(int u, int v, lli len)
```

```
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->len = len;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->len = len;
        q->next = edges[v]; edges[v] = q;
        return ;
    }
    void eval(int s)
    {
        priority_queue<pli, vector<pli>, greater<pli>> pq;
        rep(i, 0, n)
            dist[i] = infinit;
        dist[s] = 0;
        pq.push(make_pair(dist[s], s));
        while (!pq.empty()) {
            pli pr = pq.top();
            int p = pr.second;
            pq.pop();
            if (dist[p] < pr.first)
                continue;
            for (edge *ep = edges[p]; ep; ep = ep->next)
                if (dist[p] + ep->len < dist[ep->v]) {
                    dist[ep->v] = dist[p] + ep->len;
                    pq.push(make_pair(dist[ep->v], ep->v));
                }
        }
        return ;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        memclr(edges);
        return ;
    }
fm_end(graphs, dijkstra);


fm_begin(graphs, floyd_warshall):
    // @desc  Shortest Path: Floyd-Warshall
    //        suitable for multi-source multi-target positive-weight graphs
    // @complexity  Time: O(n^3), Space: O(n^2)
    // @usage  dist[i][j]: the distance from node i to j
    // @usage  add_edge(u, v, len): create directed edge
    // @usage  eval(s): calculate all distances between vertex pairs
    fm_const(int, maxn, 1010);
    fm_const(lli, infinit, 0x007f7f7f7f7f7f7fll);
    lli dist[maxn][maxn];
    int n;
    void add_edge(int u, int v, lli len)
    {
        dist[u][v] = len;
        return ;
    }
    void eval(void)
    {
        rep(k, 1, n)
            rep(i, 1, n)
                rep(j, 1, n)
                    if (dist[i][k] + dist[k][j] < dist[i][j])
                        dist[i][j] = dist[i][k] + dist[k][j];
        return ;
    }
    void init(int n)
    {
        this->n = n;
        rep(i, 1, n)
            rep(j, 1, n)
                dist[i][j] = i == j ? 0 : infinit;
        return ;
    }
fm_end(graphs, floyd_warshall);

fm_begin(graphs, tree_diameter):
    // @desc  Evaluate tree diameter (edges between farthest points)
    // @complexity  Time: O(n), Space: O(n)
    // @usage  add_edge(u, v): create edge
    // @usage  bfs(s): find farthest node with s as root
    // @usage  eval(): evaluate diameter
    // @usage  init(n): reset the graph and set vertex count as n
    fm_const(int, maxn, 100100);
    struct edge
    {
        int u, v;
        edge *next;
    };
    edge epool[2 * maxn], *edges[maxn];
    int n, ecnt;
    int depth[maxn];
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->next = edges[v]; edges[v] = q;
        return ;
    }
    int bfs(int s)
    {
        queue<int> que;
        memclr(depth);
        depth[s] = 1;
        que.push(s);
        while (!que.empty()) {
            int p = que.front();
            que.pop();
            for (edge *ep = edges[p]; ep; ep = ep->next)
                if (!depth[ep->v]) {
                    depth[ep->v] = depth[p] + 1;
                    que.push(ep->v);
                }
        }
        int maxd = 0;
        rep(i, 1, n)
            if (depth[i] > depth[maxd])
                maxd = i;
        return maxd;
    }
    int eval(void)
    {
        int p = bfs(1),
            q = bfs(p);
        return depth[q] - 1;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        memclr(edges);
        return ;
    }
fm_end(graphs, tree_diameter);


fm_begin(graphs, tree_center):
    // @desc  Evaluate tree center (when root removed the vertex with most
    //        descendants has minimum descendants possible)
    // @complexity  Time: O(n), Space: O(n)
    // @usage  add_edge(u, v): create edge
    // @usage  bfs(s): find farthest node with s as root
    // @usage  eval(): evaluate diameter
    // @usage  init(n): reset the graph and set vertex count as n
    fm_const(int, maxn, 100100);
    struct edge
    {
```

```
        int u, v;
        edge *next;
    };
    edge epool[2 * maxn], *edges[maxn];
    int n, ecnt;
    int size[maxn];
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->next = edges[v]; edges[v] = q;
        return ;
    }
    void dfs(int p, int par, int& min_p, int& min_size)
    {
        size[p] = 1;
        int maxcnt = 0;
        for (edge *ep = edges[p]; ep; ep = ep->next)
            if (ep->v != par) {
                dfs(ep->v, p, min_p, min_size);
                size[p] += size[ep->v];
                maximize(maxcnt, size[ep->v]);
            }
        maximize(maxcnt, n - size[p]);
        if (maxcnt < min_size) {
            min_p = p;
            min_size = maxcnt;
        }
        return ;
    }
    int eval(void)
    {
        int min_p = 0, min_size = n;
        dfs(1, 0, min_p, min_size);
        return min_p;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        memclr(edges);
        return ;
    }
fm_end(graphs, tree_center);


fm_begin(graphs, heavy_light_decomposition):
    // @desc  Heavy-light decomposition
    // @complexity  Time: O(n Log[n]^2), Space: O(n)
    // @warning  incompatible code style
    // 你决定设计你自己的软件包管理器。不可避免地，你要解决软件包之间的依赖
    // 问题。如果软件包A依赖软件包B，那么安装软件包A以前，必须先安装软件包B。
    // 同时，如果想要卸载软件包B，则必须卸载软件包A。现在你已经获得了所有的
    // 软件包之间的依赖关系。而且，由于你之前的工作，除0号软件包以外，在你的
    // 管理器当中的软件包都会依赖一个且仅一个软件包，而0号软件包不依赖任何一个
    // 软件包。依赖关系不存在环（若有m(m≥2)个软件包A1,A2,A3,…,Am，其中A1依赖
    // A2，A2依赖A3，A3依赖A4，……，Am-1依赖Am，而Am依赖A1，则称这m个软件包的
    // 依赖关系构成环），当然也不会有一个软件包依赖自己。
    // 现在你要为你的软件包管理器写一个依赖解决程序。根据反馈，用户希望在安装
    // 和卸载某个软件包时，快速地知道这个操作实际上会改变多少个软件包的安装状态
    // （即安装操作会安装多少个未安装的软件包，或卸载操作会卸载多少个已安装的
    // 软件包），你的任务就是实现这个部分。注意，安装一个已安装的软件包，或卸载
    // 一个未安装的软件包，都不会改变任何软件包的安装状态，即在此情况下，改变
    // 安装状态的软件包数为0。
    // 输入文件的第1行包含1个正整数n，表示软件包的总数。软件包从0开始编号。
    // 随后一行包含n-1个整数，相邻整数之间用单个空格隔开，分别表示1,2,3,…,
    // n-2,n-1号软件包依赖的软件包的编号。
    // 接下来一行包含1个正整数q，表示询问的总数。
    // 之后q行，每行1个询问。询问分为两种：
    // installx：表示安装软件包x
    // uninstallx：表示卸载软件包x
    // 你需要维护每个软件包的安装状态，一开始所有的软件包都处于未安装状态。
    // 对于每个操作，你需要输出这步操作会改变多少个软件包的安装状态，随后应用
    // 这个操作（即改变你维护的安装状态）。
    fm_const(int, maxn, 100100);
    fm_const(int, maxm, 400100);
    fm_const(int, maxlog, 17);
    static class SegmentTree
    {
    public:
        struct interval {
            int lc, rc; // Left and right (boundary) colours
            int cols; // Total consecutive colours
            void set_colour(int col) {
                lc = rc = col;
                cols = 1;
                return ; }
            interval(void) {
                lc = rc = 0;
                cols = 1; }
            interval(int col) {
                lc = rc = col;
                cols = 1; }
            interval(int l, int r, int col) {
                lc = l, rc = r, cols = col;
                return ; }
        };
        interval join(const interval& a, const interval& b) const {
            int cols = a.cols + b.cols;
            if (a.rc == b.lc) cols -= 1;
            return interval(a.lc, b.rc, cols);
        }
        struct node
        {
            node *lc, *rc;
            int lb, mb, rb, lazy;
            interval val;
        } *root, npool[maxn<<1];
        int n, ncnt;
        node* make_node(void)
        {
            node *p = &npool[++ncnt];
            p->lc = p->rc = NULL;
            p->lb = p->mb = p->rb = 0;
            p->lazy = -1;
            return p;
        }
        void dispatch_lazy(node *p)
        {
            if (p->lazy < 0 || p->lb == p->rb)
                return ;
            p->lc->lazy = p->rc->lazy = p->lazy;
            p->lc->val.set_colour(p->lazy);
            p->rc->val.set_colour(p->lazy);
            p->lazy = -1;
            return ;
        }
        void change(node *p, int l, int r, int col)
        {
            if (p->lb == l && p->rb == r) {
                p->lazy = col;
```

```
            p->val.set_colour(col);
            return ;
        }
        dispatch_lazy(p);
        if (r <= p->mb) {
            change(p->lc, l, r, col);
        } else if (l > p->mb) {
            change(p->rc, l, r, col);
        } else {
            change(p->lc, l, p->mb, col);
            change(p->rc, p->mb + 1, r, col);
        }
        p->val = join(p->lc->val, p->rc->val);
        return ;
    }
    void change(int l, int r, int col)
    {
        return this->change(root, l, r, col);
    }
    interval query(node *p, int l, int r)
    {
        if (p->lb == l && p->rb == r) {
            return p->val;
        }
        dispatch_lazy(p);
        if (r <= p->mb) {
            return query(p->lc, l, r);
        } else if (l > p->mb) {
            return query(p->rc, l, r);
        } else {
            return join(query(p->lc, l, p->mb),
                query(p->rc, p->mb + 1, r));
        }
        return interval();
    }
    interval query(int l, int r)
    {
        return this->query(root, l, r);
    }
    int query(int pos)
    {
        node *p = root;
        while (p->lb < p->rb) {
            dispatch_lazy(p);
            if (pos <= p->mb)
                p = p->lc;
            else
                p = p->rc;
        }
        return p->val.lc;
    }
    node* build_tree(int l, int r, int arr[])
    {
        node *p = make_node();
        int mid = (l + r) >> 1;
        p->lb = l; p->rb = r; p->mb = mid;
        if (p->lb == p->rb) {
            p->val = interval(arr[mid]);
        } else {
            p->lc = build_tree(l, mid, arr);
            p->rc = build_tree(mid + 1, r, arr);
            p->val = join(p->lc->val, p->rc->val);
        }
        return p;
    }
    void build(int n, int arr[])
    {
        root = build_tree(1, n, arr);
        return ;
    }
} st;
static class TreeChainPartition
{
public:
struct edge
{
    int u, v;
    edge *next;
};
int n, root, ecnt, dcnt;
int alt_arr[maxn];
edge *edges[maxn], epool[maxm];
void add_edge(int u, int v)
{
    edge *p = &epool[++ecnt],
        *q = &epool[++ecnt];
    p->u = u; p->v = v;
    q->u = v; q->v = u;
    p->next = edges[u]; edges[u] = p;
    q->next = edges[v]; edges[v] = q;
    return ;
}
int size[maxn], par[maxn], depth[maxn];
int maxch[maxn], ctop[maxn], dfn[maxn];
int jump[maxn][maxlog+1]; // Reserved for LCA
void dfs1(int p)
{
    size[p] = 1;
    for (int i = 1; i < maxlog; i++) {
        if (depth[p] < (1<<i))
            break;
        jump[p][i] = jump[jump[p][i-1]][i-1];
    }
    for (edge *ep = edges[p]; ep; ep = ep->next)
        if (ep->v != par[p]) {
            par[ep->v] = p;
            depth[ep->v] = depth[p] + 1;
            jump[ep->v][0] = p;
            dfs1(ep->v);
            size[p] += size[ep->v];
            if (size[ep->v] > size[maxch[p]])
                maxch[p] = ep->v;
        }
    return ;
}
void dfs2(int p, int chaintop)
{
    dfn[p] = ++dcnt;
    ctop[p] = chaintop;
    if (maxch[p])
        dfs2(maxch[p], chaintop);
    for (edge *ep = edges[p]; ep; ep = ep->next)
        if (depth[ep->v] == depth[p] + 1 && ep->v != maxch[p])
            dfs2(ep->v, ep->v);
    return ;
}
int lca(int x, int y)
{
    if (depth[x] < depth[y])
        swap(x, y);
    // Ensured that x is deeper than y
    int dist = depth[x] - depth[y];
    // Letting x reach the depth par y
    for (int i = 0; i < maxlog; i++)
        if (dist & (1<<i))
            x = jump[x][i];
    // Syncing ancestors
    for (int i = maxlog - 1; i >= 0; i--)
        if (jump[x][i] != jump[y][i])
            x = jump[x][i],
            y = jump[y][i];
    if (x == y)
        return x;
    return jump[x][0];
}
void __change(int x, int y, int colour)
{
    while (ctop[x] != ctop[y]) {
```

```
            st.change(dfn[ctop[x]], dfn[x], colour);
            x = jump[ctop[x]][0];
        }
        st.change(dfn[y], dfn[x], colour);
        return ;
    }
    int __query(int x, int y)
    {
        int res = 0;
        while (ctop[x] != ctop[y]) {
            int tmp = st.query(dfn[ctop[x]], dfn[x]).cols;
            res += tmp;
            if (st.query(dfn[jump[ctop[x]][0]]) ==
st.query(dfn[ctop[x]]))
                res -= 1;
            x = jump[ctop[x]][0];
        }
        int tmp = st.query(dfn[y], dfn[x]).cols;
        res += tmp;
        return res;
    }
    void change(int x, int y, int colour)
    {
        int z = lca(x, y);
        __change(x, z, colour);
        __change(y, z, colour);
        return ;
    }
    int query(int x, int y)
    {
        int z = lca(x, y);
        int res = __query(x, z)
            + __query(y, z) - 1;
        return res;
    }
    void init(int n, int arr[])
    {
        this->n = n;
        dcnt = 0;
        this->root = 1;
        // Generating DFS sequences
        depth[root] = 1;
        dfs1(root);
        dfs2(root, root);
        // Building segment tree, with minor modifications
        for (int i = 1; i <= n; i++)
            alt_arr[dfn[i]] = arr[i];
        st.build(n, alt_arr);
        return ;
    }
} graph;
int n, m;
int arr[maxn];
char str[64];
void main(void)
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", &arr[i]);
    for (int i = 1, a, b; i <= n - 1; i++) {
        scanf("%d%d", &a, &b);
        graph.add_edge(a, b);
    }
    // Building graph with integrated functions
    graph.init(n, arr);
    // Answering queries
    for (int idx = 1; idx <= m; idx++) {
        scanf("%s", str);
        int a, b, c;
        if (str[0] == 'C') {
            scanf("%d%d%d", &a, &b, &c);
            graph.change(a, b, c);
        } else if (str[0] == 'Q') {
            scanf("%d%d", &a, &b);
            int res = graph.query(a, b);
            printf("%d\n", res);
        }
    }
    // Finished
    return ;
}
fm_end(graphs, heavy_light_decomposition);


fm_begin(graphs, link_cut_tree):
    // @desc  Link-cut tree
    // @complexity  Time: O(n Log[n]^2), Space: O(n)
    // @warning  incompatible code style
    // @warning  missing pushdown and pushup functions
    // 某天，Lostmonkey发明了一种超级弹力装置，为了在他的绵羊朋友面前显
摆，他
    // 邀请小绵羊一起玩个游戏。游戏一开始，Lostmonkey在地上沿着一条直线
摆上n个
    // 装置，每个装置设定初始弹力系数ki，当绵羊达到第i个装置时，它会往
后弹ki
    // 步，达到第i+ki个装置，若不存在第i+ki个装置，则绵羊被弹飞。绵羊
想知道当它
    // 从第i个装置起步时，被弹几次后会被弹飞。为了使得游戏更有趣，
Lostmonkey可
    // 以修改某个弹力装置的弹力系数，任何时候弹力系数均为正整数。
    // 第一行包含一个整数n，表示地上有n个装置，装置的编号从0到n-1,接下
来一行有
    // n个正整数，依次为那n个装置的初始弹力系数。第三行有一个正整数m，
接下来m
    // 行每行至少有两个数i、j，若i=1，你要输出从j出发被弹几次后被弹
飞，若i=2则
    // 还会再输入一个正整数k，表示第j个弹力装置的系数被修改成k。
    fm_const(int, maxn, 200100);
    int arr_i[maxn][5];
    #define lc(_x) arr_i[_x][0]
    #define rc(_x) arr_i[_x][1]
    #define ch(_x,_y) arr_i[_x][_y]
    #define par(_x) arr_i[_x][2]
    #define size(_x) arr_i[_x][3]
    #define isroot(_x) arr_i[_x][4]
    int n;
    void update_lazy(int p)
    {
        size(p) = size(lc(p)) + 1 + size(rc(p));
        return ;
    }
    void rotate(int p)
    {
        int q = par(p), g = par(q);
        int x = rc(q) == p, y = q == rc(g);
        ch(q, x) = ch(p, !x); if (ch(q, x)) par(ch(q, x)) = q;
        ch(p, !x) = q; par(q) = p;
        par(p) = g; // if (g) ch(g, y) = p;
        if (isroot(q)) {
            isroot(p) = true;
            isroot(q) = false;
        } else {
            ch(g, y) = p;
        }
        update_lazy(q);
        update_lazy(p);
        return ;
    }
    void splay(int p)
    {
        for (int q = 0; (q = par(p)) && !isroot(p); rotate(p))
            if (par(q) && !isroot(q))
                rotate((p == rc(q)) == (q == rc(par(q))) ? q : p);
        return ;
    }
    void access(int p)
    {
        int q = 0;
```

```
        while (p) {
            splay(p);
            isroot(rc(p)) = true;
            isroot(q) = false;
            rc(p) = q;
            update_lazy(p);
            q = p, p = par(p);
        }
        return ;
    }
    void makeroot(int p)
    {
        access(p);
        splay(p);
        return ;
    }
    void link(int p, int q)
    {
        makeroot(p);
        par(lc(p)) = 0;
        isroot(lc(p)) = true;
        lc(p) = 0;
        par(p) = q;
        update_lazy(p);
        return ;
    }
    void init(int n_)
    {
        this->n = n_;
        for (int i = 1; i <= n; i++) {
            isroot(i) = true;
            size(i) = 1;
        }
        return ;
    }
    int query(int p)
    {
        makeroot(p);
        return size(lc(p)) + 1;
    }
    #undef lc
    #undef rc
    #undef ch
    #undef par
    #undef size
    #undef isroot
    void main(void)
    {
        int n, m;
        scanf("%d", &n);
        init(n);
        for (int i = 1, a; i <= n; i++) {
            scanf("%d", &a);
            if (i + a <= n)
                link(i, i + a);
        }
        scanf("%d", &m);
        for (int idx = 1; idx <= m; idx++) {
            int a, b, c;
            scanf("%d", &a);
            if (a == 1) { // To query
                scanf("%d", &b);
                b += 1; // Due to the strange marker descripted in the
problem
                printf("%d\n", query(b));
            } else if (a == 2) { // To modify
                scanf("%d%d", &b, &c);
                b += 1; // Due to the strange marker descripted in the
problem
                link(b, b+c <= n ? b+c : 0);
            }
        }
        return ;
    }
```

```
fm_end(graphs, link_cut_tree);


fm_begin(graphs, prim):
    // @desc  Minimum span tree: Prim
    //        suitable for dense graphs
    // @complexity  Time: O(n^2), Space: O(n^2)
    // @usage  dist[i][j]: direct weight between vertex i and j
    // @usage  vis[i]: true if i was visited
    // @usage  min_cost[i]: the id of the closest node to i, and is already
    //         inside the minimum span tree
    // @usage  add_edge(u, v, len): create edge
    // @usage  mst(graph): evaluate mst and store edges in graph
    // @usage  init(n): reset the graph and set vertex count as n
    fm_const(int, maxn, 1010);
    fm_const(lli, infinit, 0x007f7f7f7f7f7f7fll);
    lli dist[maxn][maxn];
    int n, vis[maxn], min_cost[maxn];
    void add_edge(int u, int v, lli len)
    {
        dist[u][v] = dist[v][u] = len;
        return ;
    }
    lli mst(fm(graphs, basic_graph)& graph)
    {
        lli min_span = 0;
        graph.init();
        rep(i, 1, n) {
            vis[i] = false;
            min_cost[i] = 1;
        }
        vis[1] = true;
        rep(i, 1, n) {
            int p = 0;
            rep(j, 1, n)
                if (!vis[j] && dist[min_cost[j]][j] < dist[min_cost[p]][p])
                    p = j;
            if (p == 0)
                break;
            min_span += dist[min_cost[p]][p];
            graph.add_edge_bi(min_cost[p], p, dist[min_cost[p]][p]);
            vis[p] = true;
            rep(j, 1, n)
                if (dist[p][j] < dist[min_cost[j]][j])
                    min_cost[j] = p;
        }
        return min_span;
    }
    void init(int n)
    {
        this->n = n;
        rep(i, 0, n)
            rep(j, 0, n)
                dist[i][j] = infinit;
        return ;
    }
fm_end(graphs, prim);


fm_begin(graphs, kruskal):
    // @desc  Minimum span tree: Kruskal
    //        suitable for sparse graphs
    // @complexity  Time: O(m Log[m]), Space: O(m)
    // @usage  edges[i]: the i-th edge
    // @usage  add_edge(u, v, len): create edge
    // @usage  mst(graph): evaluate mst and store edges in graph
    // @usage  init(n): reset the graph and set vertex count as n
    fm_const(int, maxn, 100100);
    fm_const(int, maxm, 1001000);
    struct edge
    {
        int u, v;
        lli len;
        bool operator < (const edge& b) const
```

```
        {
            return this->len < b.len;
        }
    };
    edge edges[maxm];
    int n, m;
    fm(trees, disjoint_set) djs;
    void add_edge(int u, int v, lli len)
    {
        edge *ep = &edges[++m];
        ep->u = u; ep->v = v; ep->len = len;
        return ;
    }
    lli mst(fm(graphs, basic_graph)& graph)
    {
        lli min_span = 0;
        int mst_cnt = 0;
        djs.init(n);
        sort(edges + 1, edges + m);
        rep(i, 1, m) {
            if (mst_cnt == n - 1)
                break;
            int u = edges[i].u, v = edges[i].v, len = edges[i].len,
                gu = djs.find(u), gv = djs.find(v);
            if (gu == gv)
                continue;
            djs.join(gu, gv);
            min_span += len;
            mst_cnt += 1;
            graph.add_edge_bi(u, v, len);
        }
        return min_span;
    }
    void init(int n)
    {
        this->n = n;
        m = 0;
        return ;
    }
fm_end(graphs, kruskal);


fm_begin(graphs, scc_tarjan):
    // @desc  Strongly connected components: Tarjan
    //        A scc is a subgraph such that all nodes can reach each other
in
    //        this directed graph
    // @complexity  Time: O(n+m), Space: O(n+m)
    // @usage  add_edge(u, v): creates edge
    // @usage  init(n): clears graph
    // @usage  eval(): how many scc(s) in graph
    // @usage  belong[i]: the id of the scc i belongs to
    // @usage  bsize[i]: the size of the i-th scc
    fm_const(int, maxn, 1010);
    fm_const(int, maxm, 20010);
    struct edge
    {
        int u, v;
        edge *next;
    };
    edge epool[maxm], *edges[maxn];
    int n, ecnt, dcnt, bcnt;
    stack<int> stk;
    int instk[maxn], dfn[maxn], low[maxn];
    int belong[maxn], bsize[maxn];
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt];
        p->u = u; p->v = v;
        p->next = edges[u]; edges[u] = p;
        return ;
    }
    void dfs(int p)
    {
        low[p] = dfn[p] = ++dcnt;
        stk.push(p);
        instk[p] = true;
        for (edge *ep = edges[p]; ep; ep = ep->next) {
            int q = ep->v;
            if (!dfn[q]) {
                dfs(q);
                if (low[q] < low[p])
                    low[p] = low[q];
            } else if (instk[q] && dfn[q] < low[p]) {
                low[p] = dfn[q];
            }
        }
        if (dfn[p] == low[p]) {
            bsize[++bcnt] = 0;
            int q = 0;
            do {
                q = stk.top();
                stk.pop();
                instk[q] = false;
                belong[q] = bcnt;
                bsize[bcnt] += 1;
            } while (q != p);
        }
        return ;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        memclr(edges);
        return ;
    }
    int eval(void)
    {
        while (!stk.empty())
            stk.pop();
        dcnt = bcnt = 0;
        memclr(dfn);
        memclr(low);
        memclr(instk);
        memclr(belong);
        rep(i, 1, n)
            if (!dfn[i])
                dfs(i);
        return bcnt;
    }
fm_end(graphs, scc_tarjan);


fm_begin(graphs, dcc_tarjan_v):
    // @desc  Double connected components (Vertices): Tarjan
    //        A dcc-v is a subgraph such that all nodes can reach each
other in
    //        at least two paths, different in nodes
    // @complexity  Time: O(n+m), Space: O(n+m)
    // @usage  add_edge(u, v): creates edge
    // @usage  init(n): clears graph
    // @usage  eval(): how many dcc(s) in graph
    // @usage  dcc[i]: a vector describing a dcc
    // @usage  is_cut[i]: if vertex i is a cut, a cut is a vertex such that
    //         removing it makes the graph disconnected
    fm_const(int, maxn, 1010);
    fm_const(int, maxm, 20010);
    struct edge
    {
        int u, v;
        edge *next;
    };
    edge epool[maxm], *edges[maxn];
    int n, ecnt, dcnt, bcnt;
    stack<edge*> stk;
    int instk[maxn], dfn[maxn], low[maxn];
    int belong[maxn];
```

```
    vector<int> dcc[maxn];
    bool is_cut[maxn];
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u;
        q->next = edges[v]; edges[v] = q;
        return ;
    }
    void dfs(int p, int par)
    {
        int child = 0;
        dfn[p] = low[p] = ++dcnt;
        for (edge *ep = edges[p]; ep; ep = ep->next) {
            int q = ep->v;
            if (!dfn[q]) {
                stk.push(ep);
                child += 1;
                dfs(ep->v, p);
                minimize(low[p], low[q]);
                if (dfn[p] <= low[q]) {
                    is_cut[p] = true;
                    bcnt += 1;
                    edge *eq = nullptr;
                    do {
                        eq = stk.top();
                        stk.pop();
                        if (belong[eq->u] != bcnt) {
                            belong[eq->u] = bcnt;
                            dcc[bcnt].push_back(eq->u);
                        }
                        if (belong[eq->v] != bcnt) {
                            belong[eq->v] = bcnt;
                            dcc[bcnt].push_back(eq->v);
                        }
                    } while (eq->u != p || eq->v != q);
                }
            } else if (dfn[q] < dfn[p] && q != par) {
                stk.push(ep);
                minimize(low[p], dfn[q]);
            }
        }
        if (par == 0 && child == 1)
            is_cut[p] = false;
        return ;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        memclr(edges);
        return ;
    }
    int eval(void)
    {
        while (!stk.empty())
            stk.pop();
        dcnt = bcnt = 0;
        memclr(dfn);
        memclr(low);
        memclr(instk);
        memclr(belong);
        rep(i, 1, n)
            dcc[i].clear();
        memclr(is_cut);
        rep(i, 1, n)
            if (!dfn[i])
                dfs(i, 0);
        return bcnt;
    }
fm_end(graphs, dcc_tarjan_v);
```

```
fm_begin(graphs, dcc_tarjan_e):
    // @desc  Double connected components (Edges): Tarjan
    //        A dcc-e is a subgraph such that at least two paths composed
of
    //        distinct edges exists between two arbitrary nodes
    // @complexity  Time: O(n+m), Space: O(n+m)
    // @usage  add_edge(u, v): creates edge
    // @usage  init(n): clears graph
    // @usage  eval(): how many dcc(s) in graph
    // @usage  belong[i]: the id of the dcc i belongs to
    // @usage  bsize[i]: the size of the i-th dcc
    // @usage  bridges: a vector of bridges, such that removing this edge
makes
    //         the graph disconnected
    fm_const(int, maxn, 1010);
    fm_const(int, maxm, 20010);
    struct edge
    {
        int u, v;
        bool is_bridge;
        edge *next, *rev;
    };
    edge epool[maxm], *edges[maxn];
    int n, ecnt, dcnt, bcnt;
    int dfn[maxn], low[maxn];
    int belong[maxn], bsize[maxn];
    vector<pair<int, int>> bridges;
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->is_bridge = false;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->is_bridge = false;
        q->next = edges[v]; edges[v] = q;
        p->rev = q; q->rev = p;
        return ;
    }
    void tarjan(int p, int par)
    {
        dfn[p] = low[p] = ++dcnt;
        for (edge *ep = edges[p]; ep; ep = ep->next) {
            int q = ep->v;
            if (!dfn[q]) {
                tarjan(q, p);
                minimize(low[p], low[q]);
                if (low[q] > dfn[p])
                    ep->is_bridge = ep->rev->is_bridge = true;
            } else if (dfn[q] < dfn[p] && q != par) {
                minimize(low[p], dfn[q]);
            }
        }
        return ;
    }
    void dfs(int p)
    {
        dfn[p] = true;
        belong[p] = bcnt;
        bsize[bcnt] += 1;
        for (edge *ep = edges[p]; ep; ep = ep->next) {
            if (ep->is_bridge) {
                if (ep->u < ep->v)
                    bridges.push_back(make_pair(ep->u, ep->v));
                continue;
            }
            if (!dfn[ep->v])
                dfs(ep->v);
        }
        return ;
    }
    void init(int n)
    {
```

```
        this->n = n;
        ecnt = 0;
        memclr(edges);
        return ;
    }
    int eval(void)
    {
        dcnt = bcnt = 0;
        memclr(dfn);
        memclr(low);
        memclr(belong);
        rep(i, 1, n)
            if (!dfn[i])
                tarjan(i, 0);
        memclr(dfn);
        bridges.clear();
        rep(i, 1, n)
            if (!dfn[i]) {
                bsize[++bcnt] = 0;
                dfs(i);
            }
        return bcnt;
    }
fm_end(graphs, dcc_tarjan_e);


fm_begin(graphs, 2_sat):
    // @desc  2-satisfiability, uses Tarjan engine
    // @complexity  Time: O(n+m), Space: O(n+m)
    // @usage  init(): n variables
    // @usage  constrain(c, i): unary constraint operator
    // @usage  constrain(c, i, j): binary constraint operator
    // @usage  eval(): if the situation is satisfiable
    typedef int constraint;
    fm_const(int, c_select, 1);  // (unary) choose i
    fm_const(int, c_deselect, 2);  // (unary) don't choose i
    fm_const(int, c_and, 3);  // choose i or j or both
    fm_const(int, c_xor, 4);  // i and j not chosen together
    fm_const(int, c_same, 5);  // status of i, j is same
    fm_const(int, c_diff, 6);  // status of i, j is opposite
    int n;
    fm(graphs, scc_tarjan) scc;
    void init(int n)
    {
        this->n = n;
        scc.init(2 * n);
        return ;
    }
    void constrain(constraint c, int i, int j = 0)
    {
        #define node(_x) (2 * (_x))
        #define rnode(_x) (2 * (_x) - 1)
        if (c == c_select) {
            scc.add_edge(rnode(i), node(i));
        } else if (c == c_deselect) {
            scc.add_edge(node(i), rnode(i));
        } else if (c == c_and) {
            scc.add_edge(rnode(i), node(j));
            scc.add_edge(rnode(j), node(i));
        } else if (c == c_xor) {
            scc.add_edge(node(i), rnode(j));
            scc.add_edge(node(j), rnode(i));
        } else if (c == c_same) {
            scc.add_edge(node(i), node(j));
            scc.add_edge(node(j), node(i));
            scc.add_edge(rnode(i), rnode(j));
            scc.add_edge(rnode(j), rnode(i));
        } else if (c == c_diff) {
            scc.add_edge(node(i), rnode(j));
            scc.add_edge(node(j), rnode(i));
            scc.add_edge(rnode(i), node(j));
            scc.add_edge(rnode(j), node(i));
        }
        #undef node
        #undef rnode
        return ;
    }
    bool eval(void)
    {
        scc.eval();
        rep(i, 1, n)
            if (scc.belong[2 * i] == scc.belong[2 * i - 1])
                return false;
        return true;
    }
fm_end(graphs, 2_sat);


fm_begin(graphs, dinic):
    // @desc  Max flow: Dinic
    //        The graph's maximum flow is also the minimum cost to cut the
    //        graph such that s and t becomes disconnected
    // @complexity  Time: O(n^2 m), Space: O(n+m)
    // @usage  add_edge(u, v, flow, rflow): create edge
    // @usage  add_edge(u, v, flow): create edge
    // @usage  init(n, s, t): init graph size n, source s, target t
    // @usage  eval(): evaluate maximum flow
    fm_const(int, maxn, 1010);
    fm_const(int, maxm, 20010);
    fm_const(lli, infinit, 0x007f7f7f7f7f7f7fll);
    struct edge
    {
        int u, v;
        lli flow;
        edge *next, *rev;
    };
    edge epool[maxm], *edges[maxn];
    int n, s, t, ecnt, level[maxn];
    void add_edge(int u, int v, lli flow, lli rflow)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->flow = flow;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->flow = rflow;
        q->next = edges[v]; edges[v] = q;
        p->rev = q; q->rev = p;
        return ;
    }
    void add_edge(int u, int v, lli flow)
    {
        add_edge(u, v, flow, 0);
        return ;
    }
    bool make_level(void)
    {
        memclr(level);
        queue<int> que;
        level[s] = 1;
        que.push(s);
        while (!que.empty()) {
            int p = que.front();
            que.pop();
            for (edge *ep = edges[p]; ep; ep = ep->next)
                if (ep->flow > 0 && !level[ep->v]) {
                    level[ep->v] = level[p] + 1;
                    que.push(ep->v);
                }
            if (level[t])
                return true;
        }
        return level[t] > 0;
    }
    lli find(int p, lli mn)
    {
        if (p == t)
            return mn;
        lli tmp = 0, sum = 0;
```

```
        for (edge *ep = edges[p]; ep && sum < mn; ep = ep->next)
            if (ep->flow && level[ep->v] == level[p] + 1) {
                tmp = find(ep->v, min(mn, ep->flow));
                if (tmp > 0) {
                    sum += tmp;
                    ep->flow -= tmp;
                    ep->rev->flow += tmp;
                    return tmp;
                }
            }
        if (sum == 0)
            level[p] = 0;
        return 0;
    }
    void init(int n, int s, int t)
    {
        this->n = n; this->s = s; this->t = t;
        ecnt = 0;
        memclr(edges);
        return ;
    }
    lli eval(void)
    {
        lli tmp, sum = 0;
        while (make_level()) {
            bool found = false;
            while (tmp = find(s, infinit)) {
                sum += tmp;
                found = true;
            }
            if (!found)
                break;
        }
        return sum;
    }
fm_end(graphs, dinic);


fm_begin(graphs, spfa_costflow):
    // @desc  Cost flow (Maximum flow, then minimum cost): SPFA ver.
    // @complexity  Time: O(n m^2), Space: O(n+m)
    // @usage  add_edge(u, v, flow, cost): create edge
    // @usage  init(n, s, t): init graph size n, source s, target t
    // @usage  eval(): evaluate minimum cost under maximum flow
    fm_const(int, maxn, 1010);
    fm_const(int, maxm, 20010);
    fm_const(lli, infinit, 0x007f7f7f7f7f7f7fll);
    struct edge
    {
        int u, v;
        lli flow, cost;
        edge *next, *rev;
    };
    edge epool[maxm], *edges[maxn], *from[maxn];
    int n, s, t, ecnt, inque[maxn];
    lli dist[maxn];
    typedef pair<lli, int> pli;
    void add_edge(int u, int v, lli flow, lli cost)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->flow = flow; p->cost = cost;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->flow = 0; q->cost = - cost;
        q->next = edges[v]; edges[v] = q;
        p->rev = q; q->rev = p;
        return ;
    }
    bool spfa(void)
    {
        rep(i, 1, n) {
            inque[i] = false;
            dist[i] = infinit;
            from[i] = nullptr;
        }
        priority_queue<pli, vector<pli>, greater<pli>> pq;
        inque[s] = true;
        dist[s] = 0;
        pq.push(make_pair(dist[s], s));
        while (!pq.empty()) {
            pli pr = pq.top();
            int p = pr.second;
            pq.pop();
            if (dist[p] < pr.first)
                continue;
            for (edge *ep = edges[p]; ep; ep = ep->next)
                if (ep->flow && dist[p] + ep->cost < dist[ep->v]) {
                    dist[ep->v] = dist[p] + ep->cost;
                    from[ep->v] = ep;
                    if (!inque[ep->v]) {
                        inque[ep->v] = true;
                        pq.push(make_pair(dist[ep->v], ep->v));
                    }
                }
            inque[p] = false;
        }
        return dist[t] < infinit;
    }
    void init(int n, int s, int t)
    {
        this->n = n; this->s = s; this->t = t;
        ecnt = 0;
        memclr(edges);
        return ;
    }
    lli eval(void)
    {
        lli tmp, sum = 0;
        while (spfa()) {
            tmp = infinit;
            for (edge *ep = from[t]; ep; ep = from[ep->u])
                minimize(tmp, ep->flow);
            for (edge *ep = from[t]; ep; ep = from[ep->u]) {
                ep->flow -= tmp;
                ep->rev->flow += tmp;
            }
            sum += tmp * dist[t];
        }
        return sum;
    }
fm_end(graphs, spfa_costflow);


fm_begin(graphs, zkw_costflow):
    // @desc  Cost flow (Maximum flow, then minimum cost): ZKW ver.
    // @complexity  Time: O(n m), Space: O(n+m)
    // @usage  add_edge(u, v, flow, cost): create edge
    // @usage  init(n, s, t): init graph size n, source s, target t
    // @usage  eval(): evaluate minimum cost under maximum flow
    fm_const(int, maxn, 1010);
    fm_const(int, maxm, 20010);
    fm_const(lli, infinit, 0x007f7f7f7f7f7f7fll);
    struct edge
    {
        int u, v;
        lli flow, cost;
        edge *next, *rev;
    };
    edge epool[maxm], *edges[maxn], *cursor[maxn];
    int n, s, t, ecnt, cur[maxn], vis[maxn];
    lli dist[maxn];
    void add_edge(int u, int v, lli flow, lli cost)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->flow = flow; p->cost = cost;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->flow = 0; q->cost = - cost;
```

```
        q->next = edges[v]; edges[v] = q;
        p->rev = q; q->rev = p;
        return ;
    }
    lli augment(int p, lli mn)
    {
        if (p == t)
            return mn;
        vis[p] = true;
        for (edge *ep = cursor[p]; ep; ep = ep->next)
            if (ep->flow && !vis[ep->v] && dist[ep->v] + ep->cost == dist[p]) {
                lli tmp = augment(ep->v, min(mn, ep->flow));
                if (tmp > 0) {
                    ep->flow -= tmp;
                    ep->rev->flow += tmp;
                    cursor[p] = ep;
                    return tmp;
                }
            }
        return 0;
    }
    bool mod_label(void)
    {
        lli tmp = infinit;
        rep(i, 1, n)
            if (vis[i])
                for (edge *ep = edges[i]; ep; ep = ep->next)
                    if (ep->flow && !vis[ep->v])
                        minimize(tmp, dist[ep->v] + ep->cost - dist[i]);
        if (tmp == infinit)
            return false;
        rep(i, 1, n)
            if (vis[i]) {
                vis[i] = false;
                dist[i] += tmp;
            }
        return true;
    }
    void init(int n, int s, int t)
    {
        this->n = n; this->s = s; this->t = t;
        ecnt = 0;
        memclr(edges);
        return ;
    }
    lli eval(void)
    {
        lli tmp, sum = 0;
        do {
            rep(i, 1, n)
                cursor[i] = edges[i];
            while (tmp = augment(s, infinit)) {
                sum += tmp * dist[s];
                memclr(vis);
            }
        } while (mod_label());
        return sum;
    }
fm_end(graphs, zkw_costflow);


fm_begin(graphs, hungary_match):
    // @desc  Bipartite maximum match: Hungary algorithm
    // @complexity  Time: O(n m), Space: O(n+m)
    // @usage  add_edge(u, v): create edge
    // @usage  init(n): init graph size n
    // @usage  eval(): evaluate how many matches between the bipartite graph
    //         can be made (how many pairs)
    fm_const(int, maxn, 1010);
    fm_const(int, maxm, 20010);
    struct edge
    {
        int u, v;
        edge *next;
    };
    edge epool[maxm], *edges[maxn];
    int n, ecnt, from[maxn], vis[maxn];
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt];
        p->u = u; p->v = v;
        p->next = edges[u]; edges[u] = p;
        return ;
    }
    bool find(int p)
    {
        for (edge *ep = edges[p]; ep; ep = ep->next)
            if (!vis[ep->v]) {
                vis[ep->v] = true;
                if (!from[ep->v] || find(from[ep->v])) {
                    from[ep->v] = p;
                    return true;
                }
            }
        return false;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        memclr(edges);
        return ;
    }
    int eval(void)
    {
        int sum = 0;
        rep(i, 1, n) {
            memclr(vis);
            if (!from[i])
                sum += (int)(find(i));
        }
        return sum;
    }
fm_end(graphs, hungary_match);


fm_begin(graphs, bron_kerbosch):
    // @desc  Max clique: Bron-Kerbosch algorithm
    //        Largest independent set
    // @complexity  Time: O(3^(n/3)), Space: O(n^2)
    // @usage  add_edge(u, v): create edge
    // @usage  init(n): clear graph
    // @usage  eval(): yields size of max clique
    // @usage  clique[i]: the i-th vertex in its max clique
    // @usage  if largest independent set (largest subgraph that are not
    //         connected to each other) is required, send its supplementary
    //         graph to this algorithm and the max clique is the result
    fm_const(int, maxn, 1010);
    bool edge[maxn][maxn];
    int n, cnt[maxn], vis[maxn], clique[maxn];
    void add_edge(int u, int v)
    {
        edge[u][v] = edge[v][u] = true;
        return ;
    }
    bool dfs(int p, int pos, int& res)
    {
        rep(i, p + 1, n) {
            if (cnt[i] + pos <= res)
                return false;
            if (edge[p][i]) {
                int j = 0;
                for (; j < pos; j++)
                    if (!edge[i][vis[j]])
                        break;
                if (j == pos) {
```

```
                    vis[j] = i;
                    if (dfs(i, pos + 1, res))
                        return true;
                }
            }
        }
        if (pos > res) {
            res = pos;
            rep(i, 0, res - 1)
                clique[i + 1] = vis[i];
            return true;
        }
        return false;
    }
    void init(int n)
    {
        this->n = n;
        memclr(edge);
        memclr(cnt);
        memclr(vis);
        memclr(clique);
        return ;
    }
    int eval(void)
    {
        int res = -1;
        rep_(i, n, 1) {
            vis[0] = i;
            dfs(i, 1, res);
            cnt[i] = res;
        }
        return res;
    }
fm_end(graphs, bron_kerbosch);


// @desc  Kuangbin's computational geometry templates
// @warning  these templates are expected to run smoothly, but not expected
//           to pass compiler here
namespace kuangbin
{
namespace chapter_1_x
{
// @chapter 1.1 Point 定义
const double eps = 1e-8;
const double PI = acos(-1.0);
int sgn(double x)
{
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct point
{
    double x,y;
    point() {}
    point(double _x,double _y)
    {
        x = _x;
        y = _y;
    }
    point operator -(const point &b)const
    {
        return point(x - b.x,y - b.y);
    }
//叉积
    double operator ^(const point &b)const
    {
        return x*b.y - y*b.x;
    }
//点积
    double operator *(const point &b)const
    {
        return x*b.x + y*b.y;
    }
//绕原点旋转角度B（弧度值），后x,y的变化
    void transXY(double B)
    {
        double tx = x,ty = y;
        x = tx*cos(B) - ty*sin(B);
        y = tx*sin(B) + ty*cos(B);
    }
    void input()
    {
        scanf("%lf%lf",&x,&y);
    }
};
// @chapter 1.2 Line 定义
struct Line
{
    point s,e;
    Line() {}
    Line(point _s,point _e)
    {
        s = _s;
        e = _e;
    }
//两直线相交求交点
//第一个值为0表示直线重合，为1表示平行，为0表示相交,为2是相交
//只有第一个值为2时，交点才有意义
    pair<int,point> operator &(const Line &b)const
    {
        point res = s;
        if(sgn((s-e)^(b.s-b.e)) == 0)
        {
            if(sgn((s-b.e)^(b.s-b.e)) == 0)
                return make_pair(0,res);//重合
            else return make_pair(1,res);//平行
        }
        double t = ((s-b.s)^(b.s-b.e))/((s-e)^(b.s-b.e));
        res.x += (e.x-s.x)*t;
        res.y += (e.y-s.y)*t;
        return make_pair(2,res);
    }
};
// @chapter 1.3 两点间距离
//*两点间距离
double dist(point a,point b)
{
    return sqrt((a-b)*(a-b));
}
// @chapter 1.4 判断 线段相交
bool inter(Line l1,Line l2)
{
    return
        max(l1.s.x,l1.e.x) >= min(l2.s.x,l2.e.x) &&
        max(l2.s.x,l2.e.x) >= min(l1.s.x,l1.e.x) &&
        max(l1.s.y,l1.e.y) >= min(l2.s.y,l2.e.y) &&
        max(l2.s.y,l2.e.y) >= min(l1.s.y,l1.e.y) &&
        sgn((l2.s-l1.e)^(l1.s-l1.e))*sgn((l2.e-l1.e)^(l1.s-l1.e)) <= 0 &&
        sgn((l1.s-l2.e)^(l2.s-l2.e))*sgn((l1.e-l2.e)^(l2.s-l2.e)) <= 0;
}
// @chapter 1.5 判断 直线和线段相交
//判断直线和线段相交
bool Seg_inter_line(Line l1,Line l2) //判断直线l1和线段l2是否相交
{
    return sgn((l2.s-l1.e)^(l1.s-l1.e))*sgn((l2.e-l1.e)^(l1.s-l1.e)) <= 0;
}
// @chapter 1.6 点到直线距离
//点到直线距离
//返回为result,是点到直线最近的点
point PointToLine(point P,Line L)
{
    point result;
    double t = ((P-L.s)*(L.e-L.s))/((L.e-L.s)*(L.e-L.s));
    result.x = L.s.x + (L.e.x-L.s.x)*t;
    result.y = L.s.y + (L.e.y-L.s.y)*t;
    return result;
```

```
}
// @chapter 1.7 点到线段距离
point NearestPointToLineSeg(point P,Line L)
{
    point result;
    double t = ((P-L.s)*(L.e-L.s))/((L.e-L.s)*(L.e-L.s));
    if(t >= 0 && t <= 1)
    {
        result.x = L.s.x + (L.e.x - L.s.x)*t;
        result.y = L.s.y + (L.e.y - L.s.y)*t;
    }
    else
    {
        if(dist(P,L.s) < dist(P,L.e))
            result = L.s;
        else result = L.e;
    }
    return result;
}
// @chapter 1.8 计算多边形面积
//计算多边形面积
//点的编号从0~n-1
double CalcArea(point p[],int n)
{
    double res = 0;
    for(int i = 0; i < n; i++)
        res += (p[i]^p[(i+1)%n])/2;
    return fabs(res);
}
// @chapter 1.9 判断点在线段上
//*判断点在线段上
bool OnSeg(point P,Line L)
{
    return
        sgn((L.s-P)^(L.e-P)) == 0 &&
        sgn((P.x - L.s.x) * (P.x - L.e.x)) <= 0 &&
        sgn((P.y - L.s.y) * (P.y - L.e.y)) <= 0;
}
// @chapter 1.10 判断点在凸多边形内
//*判断点在凸多边形内
//点形成一个凸包，而且按逆时针排序（如果是顺时针把里面的<0改为>0）
//点的编号:0~n-1
//返回值：
//-1:点在凸多边形外
//0:点在凸多边形边界上
//1:点在凸多边形内
int inConvexPoly(point a,point p[],int n)
{
    for(int i = 0; i < n; i++)
    {
        if(sgn((p[i]-a)^(p[(i+1)%n]-a)) < 0)return -1;
        else if(OnSeg(a,Line(p[i],p[(i+1)%n])))return 0;
    }
    return 1;
}
// @chapter 1.11 判断点在任意多边形内
//*判断点在任意多边形内
//射线法，poly[]的顶点数要大于等于3,点的编号0~n-1
//返回值
//-1:点在凸多边形外
//0:点在凸多边形边界上
//1:点在凸多边形内
int inPoly(point p,point poly[],int n)
{
    int cnt;
    Line ray,side;
    cnt = 0;
    ray.s = p;
    ray.e.y = p.y;
    ray.e.x = -100000000000.0;//-INF,注意取值防止越界
    for(int i = 0; i < n; i++)
    {
        side.s = poly[i];
        side.e = poly[(i+1)%n];
        if(OnSeg(p,side))return 0;
//如果平行轴则不考虑
        if(sgn(side.s.y - side.e.y) == 0)
            continue;
        if(OnSeg(side.s,ray))
        {
            if(sgn(side.s.y - side.e.y) > 0)cnt++;
        }
        else if(OnSeg(side.e,ray))
        {
            if(sgn(side.e.y - side.s.y) > 0)cnt++;
        }
        else if(inter(ray,side))
            cnt++;
    }
    if(cnt % 2 == 1)return 1;
    else return -1;
}
// @chapter 1.12 判断多边形
//判断凸多边形
//允许共线边
//点可以是顺时针给出也可以是逆时针给出
//点的编号1~n-1
bool isconvex(point poly[],int n)
{
    bool s[3];
    memset(s,false,sizeof(s));
    for(int i = 0; i < n; i++)
    {
        s[sgn( (poly[(i+1)%n]-poly[i])^(poly[(i+2)%n]-poly[i]) )+1] = true;
        if(s[0] && s[2])return false;
    }
    return true;
}
// @chapter 1.13 简单极角排序
const int maxn=55;
point List[maxn];
bool _cmp(point p1,point p2)
{
    double tmp = (p1-List[0])^(p2-List[0]);
    if(sgn(tmp) > 0)return true;
    else if(sgn(tmp) == 0 && sgn(dist(p1,List[0]) - dist(p2,List[0])) <= 0)
        return true;
    else return false;
}
// sort(List+1,List+n,_cmp);
};
using namespace chapter_1_x;
namespace chapter_2_x
{
// @chapter 2.1 凸包
/*
* 求凸包，Graham算法
* 点的编号0~n-1
* 返回凸包结果Stack[0~top-1]为凸包的编号
*/
const int MAXN = 1010;
point List[MAXN];
int Stack[MAXN],top;
//相对于List[0]的极角排序
bool _cmp(point p1,point p2)
{
    double tmp = (p1-List[0])^(p2-List[0]);
    if(sgn(tmp) > 0)return true;
    else if(sgn(tmp) == 0 && sgn(dist(p1,List[0]) - dist(p2,List[0])) <= 0)
        return true;
    else return false;
}
void Graham(int n)
{
    point p0;
    int k = 0;
    p0 = List[0];
//找最下边的一个点
```

```
    for(int i = 1; i < n; i++)
    {
        if( (p0.y > List[i].y) || (p0.y == List[i].y && p0.x > List[i].x) )
        {
            p0 = List[i];
            k = i;
        }
    }
    swap(List[k],List[0]);
    sort(List+1,List+n,_cmp);
    if(n == 1)
    {
        top = 1;
        Stack[0] = 0;
        return;
    }
    if(n == 2)
    {
        top = 2;
        Stack[0] = 0;
        Stack[1] = 1;
        return ;
    }
    Stack[0] = 0;
    Stack[1] = 1;
    top = 2;
    for(int i = 2; i < n; i++)
    {
        while(top > 1 &&
                sgn((List[Stack[top-1]]-List[Stack[top-2]])^
                    (List[i]-List[Stack[top-2]])) <=
                0)top--;
        Stack[top++] = i;
    }
}
};
namespace chapter_3_x
{
// @chapter 3.1 平面最近点对
#include <stdio.h>
#include <string.h>
#include <algorithm>
#include <iostream>
#include <math.h>
using namespace std;
const double eps = 1e-6;
const int MAXN = 100010;
const double INF = 1e20;
struct Point
{
    double x,y;
};
double dist(Point a,Point b)
{
    return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
Point p[MAXN];
Point tmpt[MAXN];
bool cmpxy(Point a,Point b)
{
    if(a.x != b.x)return a.x < b.x;
    else return a.y < b.y;
}
bool cmpy(Point a,Point b)
{
    return a.y < b.y;
}
double Closest_Pair(int left,int right)
{
    double d = INF;
    if(left == right)return d;
    if(left + 1 == right)
        return dist(p[left],p[right]);
    int mid = (left+right)/2;
    double d1 = Closest_Pair(left,mid);
    double d2 = Closest_Pair(mid+1,right);
    d = min(d1,d2);
    int k = 0;
    for(int i = left; i <= right; i++)
    {
        if(fabs(p[mid].x - p[i].x) <= d)
            tmpt[k++] = p[i];
    }
    sort(tmpt,tmpt+k,cmpy);
    for(int i = 0; i <k; i++)
    {
        for(int j = i+1; j < k && tmpt[j].y - tmpt[i].y < d; j++)
        {
            d = min(d,dist(tmpt[i],tmpt[j]));
        }
    }
    return d;
}
int main()
{
    int n;
    while(scanf("%d",&n)==1 && n)
    {
        for(int i = 0; i < n; i++)
            scanf("%lf%lf",&p[i].x,&p[i].y);
        sort(p,p+n,cmpxy);
        printf("%.2lf\n",Closest_Pair(0,n-1)/2);
    }
    return 0;
}
};
namespace chapter_4_1
{
// @chapter 4.1 旋转卡壳 / 平面最远点对
struct Point
{
    int x,y;
    Point(int _x = 0,int _y = 0)
    {
        x = _x;
        y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x, y - b.y);
    }
    int operator ^(const Point &b)const
    {
        return x*b.y - y*b.x;
    }
    int operator *(const Point &b)const
    {
        return x*b.x + y*b.y;
    }
    void input()
    {
        scanf("%d%d",&x,&y);
    }
};
//距离的平方
int dist2(Point a,Point b)
{
    return (a-b)*(a-b);
}
//******二维凸包, int***********
const int MAXN = 50010;
Point list[MAXN];
int Stack[MAXN],top;
bool _cmp(Point p1,Point p2)
{
    int tmp = (p1-list[0])^(p2-list[0]);
    if(tmp > 0)return true;
    else if(tmp == 0 && dist2(p1,list[0]) <= dist2(p2,list[0]))
```

```
        return true;
    else return false;
}
void Graham(int n)
{
    Point p0;
    int k = 0;
    p0 = list[0];
    for(int i = 1; i < n; i++)
        if(p0.y > list[i].y || (p0.y == list[i].y && p0.x > list[i].x))
        {
            p0 = list[i];
            k = i;
        }
    swap(list[k],list[0]);
    sort(list+1,list+n,_cmp);
    if(n == 1)
    {
        top = 1;
        Stack[0] = 0;
        return;
    }
    if(n == 2)
    {
        top = 2;
        Stack[0] = 0;
        Stack[1] = 1;
        return;
    }
    Stack[0] = 0;
    Stack[1] = 1;
    top = 2;
    for(int i = 2; i < n; i++)
    {
        while(top > 1 &&
                ((list[Stack[top-1]]-list[Stack[top-2]])^
                        (list[i]-list[Stack[top-2]])) <= 0)
            top--;
        Stack[top++] = i;
    }
}
//旋转卡壳，求两点间距离平方的最大值
int rotating_calipers(Point p[],int n)
{
    int ans = 0;
    Point v;
    int cur = 1;
    for(int i = 0; i < n; i++)
    {
        v = p[i]-p[(i+1)%n];
        while((v^(p[(cur+1)%n]-p[cur])) < 0)
            cur = (cur+1)%n;
        ans =
max(ans,max(dist2(p[i],p[cur]),dist2(p[(i+1)%n],p[(cur+1)%n])));
    }
    return ans;
}
Point p[MAXN];
int main()
{
    int n;
    while(scanf("%d",&n) == 1)
    {
        for(int i = 0; i < n; i++)list[i].input();
        Graham(n);
        for(int i = 0; i < top; i++)p[i] = list[Stack[i]];
        printf("%d\n",rotating_calipers(p,top));
    }
    return 0;
}
};
namespace chapter_4_2
{
// @chapter 4.2 旋转卡壳计算平面点集最大三角形面积
//旋转卡壳计算平面点集最大三角形面积
int rotating_calipers(point p[],int n)
{
    int ans = 0;
    point v;
    for(int i = 0; i < n; i++)
    {
        int j = (i+1)%n;
        int k = (j+1)%n;
        while(j != i && k != i)
        {
            ans = max(ans,int(abs((p[j]-p[i])^(p[k]-p[i]))));
            while( ((p[i]-p[j])^(p[(k+1)%n]-p[k])) < 0 )
                k = (k+1)%n;
            j = (j+1)%n;
        }
    }
    return ans;
}
point p[10010];
using namespace chapter_3_x;
using namespace chapter_4_3;
int main()
{
    int n;
    while(scanf("%d",&n) == 1)
    {
        if(n == -1)break;
        for(int i = 0; i < n; i++)list[i].input();
        Graham(n);
        for(int i = 0; i < top; i++)p[i] = list[Stack[i]];
        printf("%.2f\n",(double)rotating_calipers(p,top)/2);
    }
    return 0;
}
};
namespace chapter_4_3
{
// @chapter 4.3 求解两凸包最小距离
const double eps = 1e-8;
int sgn(double x)
{
    if(fabs(x) < eps)return 0;
    if(x < 0)return -1;
    else return 1;
}
struct Point
{
    double x,y;
    Point(double _x = 0,double _y = 0)
    {
        x = _x;
        y = _y;
    }
    Point operator -(const Point &b)const
    {
        return Point(x - b.x, y - b.y);
    }
    double operator ^(const Point &b)const
    {
        return x*b.y - y*b.x;
    }
    double operator *(const Point &b)const
    {
        return x*b.x + y*b.y;
    }
    void input()
    {
        scanf("%lf%lf",&x,&y);
    }
};
struct Line
{
    Point s,e;
```

```
    Line() {}
    Line(Point _s,Point _e)
    {
        s = _s;
        e = _e;
    }
};
//两点间距离
double dist(Point a,Point b)
{
    return sqrt((a-b)*(a-b));
}
//点到线段的距离，返回点到线段最近的点
Point NearestPointToLineSeg(Point P,Line L)
{
    Point result;
    double t = ((P-L.s)*(L.e-L.s))/((L.e-L.s)*(L.e-L.s));
    if(t >=0 && t <= 1)
    {
        result.x = L.s.x + (L.e.x - L.s.x)*t;
        result.y = L.s.y + (L.e.y - L.s.y)*t;
    }
    else
    {
        if(dist(P,L.s) < dist(P,L.e))
            result = L.s;
        else result = L.e;
    }
    return result;
}
/*
* 求凸包，Graham算法
* 点的编号0~n-1
* 返回凸包结果Stack[0~top-1]为凸包的编号
*/const int MAXN = 10010;
Point list[MAXN];
int Stack[MAXN],top;
//相对于list[0]的极角排序
bool _cmp(Point p1,Point p2)
{
    double tmp = (p1-list[0])^(p2-list[0]);
    if(sgn(tmp) > 0)return true;
    else if(sgn(tmp) == 0 && sgn(dist(p1,list[0]) - dist(p2,list[0])) <= 0)
        return true;
    else return false;
}
void Graham(int n)
{
    Point p0;
    int k = 0;
    p0 = list[0];
//找最下边的一个点
    for(int i = 1; i < n; i++)
    {
        if( (p0.y > list[i].y) || (p0.y == list[i].y && p0.x > list[i].x) )
        {
            p0 = list[i];
            k = i;
        }
    }
    swap(list[k],list[0]);
    sort(list+1,list+n,_cmp);
    if(n == 1)
    {
        top = 1;
        Stack[0] = 0;
        return;
    }
    if(n == 2)
    {
        top = 2;
        Stack[0] = 0;
        Stack[1] = 1;
        return ;
    }
    Stack[0] = 0;
    Stack[1] = 1;
    top = 2;
    for(int i = 2; i < n; i++)
    {
        while(top > 1 &&
                sgn((list[Stack[top-1]]-list[Stack[top-2]])^
                    (list[i]-list[Stack[top-2]])) <=
                0)
            top--;
        Stack[top++] = i;
    }
}
//点p0到线段p1p2的距离
double pointtoseg(Point p0,Point p1,Point p2)
{
    return dist(p0,NearestPointToLineSeg(p0,Line(p1,p2)));
}//平行线段p0p1和p2p3的距离
double dispallseg(Point p0,Point p1,Point p2,Point p3)
{
    double ans1 = min(pointtoseg(p0,p2,p3),pointtoseg(p1,p2,p3));
    double ans2 = min(pointtoseg(p2,p0,p1),pointtoseg(p3,p0,p1));
    return min(ans1,ans2);
}
//得到向量a1a2和b1b2的位置关系
double Get_angle(Point a1,Point a2,Point b1,Point b2)
{
    return (a2-a1)^(b1-b2);
}
double rotating_calipers(Point p[],int np,Point q[],int nq)
{
    int sp = 0, sq = 0;
    for(int i = 0; i < np; i++)
        if(sgn(p[i].y - p[sp].y) < 0)
            sp = i;
    for(int i = 0; i < nq; i++)
        if(sgn(q[i].y - q[sq].y) > 0)
            sq = i;
    double tmp;
    double ans = dist(p[sp],q[sq]);
    for(int i = 0; i < np; i++)
    {
        while(sgn(tmp = Get_angle(p[sp],p[(sp+1)%np],q[sq],q[(sq+1)%nq])) <
0)
            sq = (sq+1)%nq;
        if(sgn(tmp) == 0)
            ans =
min(ans,dispallseg(p[sp],p[(sp+1)%np],q[sq],q[(sq+1)%nq]));
        else ans = min(ans,pointtoseg(q[sq],p[sp],p[(sp+1)%np]));
        sp = (sp+1)%np;
    }
    return ans;
}
double solve(Point p[],int n,Point q[],int m)
{
    return min(rotating_calipers(p,n,q,m),rotating_calipers(q,m,p,n));
}
Point p[MAXN],q[MAXN];
int main()
{
    int n,m;
    while(scanf("%d%d",&n,&m) == 2)
    {
        if(n == 0 && m == 0)break;
        for(int i = 0; i < n; i++)
            list[i].input();
        Graham(n);
        for(int i = 0; i < top; i++)
            p[i] = list[i];
        n = top;
        for(int i = 0; i < m; i++)
            list[i].input();
        Graham(m);
```

```
        for(int i = 0; i < top; i++)
            q[i] = list[i];
        m = top;
        printf("%.4f\n",solve(p,n,q,m));
    }
    return 0;
}
};
namespace chapter_5_1
{
// @chapter 5.1 半平面交
const double eps = 1e-8;
const double PI = acos(-1.0);
int sgn(double x)
{
    if(fabs(x) < eps) return 0;
    if(x < 0) return -1;
    else return 1;
}
struct point
{
    double x,y;
    point() {}
    point(double _x,double _y)
    {
        x = _x;
        y = _y;
    }
    point operator -(const point &b)const
    {
        return point(x - b.x, y - b.y);
    }
    double operator ^(const point &b)const
    {
        return x*b.y - y*b.x;
    }
    double operator *(const point &b)const
    {
        return x*b.x + y*b.y;
    }
};
struct Line
{
    point s,e;
    double k;
    Line() {}
    Line(point _s,point _e)
    {
        s = _s;
        e = _e;
        k = atan2(e.y - s.y,e.x - s.x);
    }
    point operator &(const Line &b)const
    {
        point res = s;
        double t = ((s - b.s)^(b.s - b.e))/((s - e)^(b.s - b.e));
        res.x += (e.x - s.x)*t;
        res.y += (e.y - s.y)*t;
        return res;
    }
};
//半平面交，直线的左边代表有效区域
//这个好像和给出点的顺序有关
bool HPIcmp(Line a,Line b)
{
    if(fabs(a.k - b.k) > eps)return a.k < b.k;
    return ((a.s - b.s)^(b.e - b.s)) < 0;
}
Line Q[110];
//第一个位代表半平面交的直线，第二个参数代表直线条数，第三个参数是相
交以后把
//所得点压栈，第四个参数是栈有多少个元素
void HPI(Line line[], int n, point res[], int &resn)
{
    int tot = n;
    sort(line,line+n,HPIcmp);
    tot = 1;
    for(int i = 1; i < n; i++)
        if(fabs(line[i].k - line[i-1].k) > eps)
            line[tot++] = line[i];
    int head = 0, tail = 1;
    Q[0] = line[0];
    Q[1] = line[1];
    resn = 0;
    for(int i = 2; i < tot; i++)
    {
        if(fabs((Q[tail].e-Q[tail].s)^(Q[tail-1].e-Q[tail-1].s)) < eps ||
                fabs((Q[head].e-Q[head].s)^(Q[head+1].e-Q[head+1].s)) <
eps)
            return;
        while(head < tail && (((Q[tail]&Q[tail-1]) -
                               line[i].s)^(line[i].e-line[i].s)) > eps)
            tail--;
        while(head < tail && (((Q[head]&Q[head+1]) -
                               line[i].s)^(line[i].e-line[i].s)) > eps)
            head++;
        Q[++tail] = line[i];
    }
    while(head < tail && (((Q[tail]&Q[tail-1]) -
                           Q[head].s)^(Q[head].e-Q[head].s)) > eps)
        tail--;
    while(head < tail && (((Q[head]&Q[head-1]) -
                           Q[tail].s)^(Q[tail].e-Q[tail].e)) > eps)
        head++;
    if(tail <= head + 1)return;
    for(int i = head; i < tail; i++)
        res[resn++] = Q[i]&Q[i+1];
    if(head < tail - 1)
        res[resn++] = Q[head]&Q[tail];
}
};
namespace chapter_6_x_7_x
{
// @chapter 6.1 三点求圆心坐标
//过三点求圆心坐标
Point waixin(Point a,Point b,Point c)
{
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1*a1 + b1*b1)/2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2*a2 + b2*b2)/2;
    double d = a1*b2 - a2*b1;
    return Point(a.x + (c1*b2 - c2*b1)/d, a.y + (a1*c2 -a2*c1)/d);
}
// @chapter 7.1 求两圆相交的面积
//两个圆的公共部分面积
double Area_of_overlap(Point c1,double r1,Point c2,double r2)
{
    double d = dist(c1,c2);
    if(r1 + r2 < d + eps)return 0;
    if(d < fabs(r1 - r2) + eps)
    {
        double r = min(r1,r2);
        return PI*r*r;
    }
    double x = (d*d + r1*r1 - r2*r2)/(2*d);
    double t1 = acos(x / r1);
    double t2 = acos((d - x)/r2);
    return r1*r1*t1 + r2*r2*t2 - d*r1*sin(t1);
}
};
};


int main(int argc, char** argv)
{
    return 0;
}
```

Contents

2-SAT 问题

- 题目：poj3678

- 依赖：

模板描述

给定 n 个布尔变量 $a_1, a_2, \dots, a_n$， m 个限定条件 $limit(f, i, j, v)$ 限制 $f(a_i, a_j) = v$。求该组限制条件是否可以达成，并给出一组解。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$

算法简述

将每个布尔变量拆成两个点 a_i^p, a_i^n，分别代表点 i 的正值和负值。每一个布尔操作的断言 $f(a_i, a_j) = v$ 都可以转化成某两个拆开的点后的连接。在程序内表现为 `constrain(p, 0, q, 0)`，代表若变量 $a_p = 0$ 则 $a_q = 1$。跑一遍强连通分量，若一个点拆开的两个点在一个强连通分量中则出现矛盾，否则 dfs 所有点即可获得一组解。所有可能解的总数为 2^{bcnt}，其中 $bcnt$ 为强连通分量的个数。

调用方法

- `int maxn`：点的数量上限。
- `void set_true(int p)`：$a_p = 1$
- `void set_false(int p)`：$a_p = 0$
- `void require_and(int p, int q)`：$a_p \land a_q = 1$
- `void require_or(int p, int q)`：$a_p \lor a_q = 1$
- `void require_nand(int p, int q)`：$a_p \land a_q = 0$
- `void require_nor(int p, int q)`：$a_p \lor a_q = 0$
- `void require_xor(int p, int q)`：$a_p \oplus a_q = 1$
- `void require_xnor(int p, int q)`：$a_p \oplus a_q = 0$
- `bool eval(int[] res)`：求解限制组，结果存入 res，若无解返回 `false`。
- `void init(int n)`：初始化点数为 n 的限制组。

```
class TwoSAT
{
public:
    int n;
    #define constrain(_p, _vp, _q, _vq)  \
            graph.add_edge(2 * (_p) - (_vp), 2 * (_q) - (_vq))
    void set_true(int p) {
```

```
    constrain(p, 0, p, 1); }
void set_false(int p) {
    constrain(p, 1, p, 0); }
void require_and(int p, int q) {
    constrain(p, 0, p, 1);
    constrain(q, 0, q, 1); }
void require_or(int p, int q) {
    constrain(p, 0, q, 1);
    constrain(q, 0, p, 1); }
void require_nand(int p, int q) {
    constrain(p, 1, q, 0);
    constrain(q, 1, p, 0); }
void require_nor(int p, int q) {
    constrain(p, 1, p, 0);
    constrain(q, 1, q, 0); }
void require_xor(int p, int q) {
    constrain(p, 1, q, 0);
    constrain(q, 1, p, 0);
    constrain(p, 0, q, 1);
    constrain(q, 0, p, 1); }
void require_xnor(int p, int q) {
    constrain(p, 1, q, 1);
    constrain(q, 1, p, 1);
    constrain(p, 0, q, 0);
    constrain(q, 0, p, 0); }
void require_eq(int p, int q) {
    require_xnor(p, q); }
void require_neq(int p, int q) {
    require_xor(p, q); }
#undef constrain
void dfs(int p, int res[])
{
    int id = (p + 1) / 2;
    if (res[id] != -1)
        return ;
    res[id] = 2 * id - p;
    for (auto* ep = graph.edges[p]; ep; ep = ep->next)
        dfs(ep->v, res);
    return ;
}
bool eval(int res[])
{
    graph.eval();
    rep(i, 1, n)
        if (graph.belong[2 * i] == graph.belong[2 * i - 1])
            return false;
    rep(i, 1, n)
        res[i] = -1;
    rep(i, 1, n)
        if (res[i] == -1)
            dfs(2 * i, res);
    return true;
}
void init(int n)
```

```
    {
        this->n = n;
        graph.init(2 * n);
        return ;
    }
} tsat;
```

Bron-Kerbosch 最大团算法

- 题目：poj1419
- 依赖：

模板描述

给定 n 个点 m 条边的无向图 $G=(V,E)$，求无向图的最大全连通分量。

最大独立集：取 $V'\subseteq V$ 且 $|V'|=k$，同时 V' 内点两两无连接。

最大团：取 $V'\subseteq V$ 且 $|V'|=k$，同时 V' 内点两两相连，也作全连通分量。

复杂度

- Space:
  - Worst Case: $O(n^2)$
  - Amortized: $O(n^2)$
  - Best Case: $O(n^2)$
- Time:
  - Worst Case: $O(3^{\frac{n}{3}})$
  - Amortized: $O(3^{\frac{n}{3}})$
  - Best Case: $O(n^2)$

算法简述

一个无向图的最大独立集为其补图的最大团。

补图：$G'=(V,U\backslash E)$ 是 G 的补图，即边联通状态完全相反。

暴力搜索所有团，适当剪枝。

一个图的最大团数量不会超过 $3^{\frac{n}{3}}$ 个。

调用方法

- `int maxn`：点的数量上限。
- `void add_edge(int u, int v)`：添加一条 $u\leftrightarrow v$ 的有向边。
- `void remove_edge(int u, int v)`：删除原来在图中的一条 $u\leftrightarrow v$ 的无向边。
- `int[] eval()`：计算该无向图的最大团大小，及其点构成。

- `void init(int n, bool state)`：对任意的边 (u, v)，初始其状态为 $state$ 。

```
class BronKerbosch
{
public:
    int n;
    bool edge[maxn][maxn];
    bool vis[maxn];
    void add_edge(int u, int v)
    {
        edge[u][v] = edge[v][u] = true;
        return ;
    }
    void remove_edge(int u, int v)
    {
        edge[u][v] = edge[v][u] = false;
        return ;
    }
    void dfs(int p, int& curn, int& bestn, vector<int>& res)
    {
        if (p > n) {
            res.clear();
            rep(i, 1, n)
                if (vis[i])
                    res.push_back(i);
            bestn = curn;
            return ;
        }
        bool flag = true;
        rep(i, 1, p - 1)
            if (vis[i] && !edge[i][p]) {
                flag = false;
                break;
            }
        if (flag) {
            curn += 1;
            vis[p] = true;
            dfs(p + 1, curn, bestn, res);
            curn -= 1;
            vis[p] = false;
        }
        if (curn + n - p > bestn)
            dfs(p + 1, curn, bestn, res);
        return ;
    }
    vector<int> eval(void)
    {
        int curn = 0, bestn = 0;
        vector<int> res;
        dfs(1, curn, bestn, res);
        return res;
    }
```

```
    void init(int n, bool state = false)
    {
        this->n = n;
        rep(i, 1, n)
            rep(j, 1, n)
                edge[i][j] = state;
        memclr(vis, n);
        return ;
    }
} graph;
```

中国剩余定理

- 题目：
- 依赖：euclid_gcd

模板描述

给定以下方程组，求 x 的最小非负整数解：

$$\begin{cases} x \equiv a_1 \mathrm{mod} m_1 \\ x \equiv a_2 \mathrm{mod} m_2 \\ x \equiv a_3 \mathrm{mod} m_3 \\ \dots \\ x \equiv a_n \mathrm{mod} m_n \end{cases}$$

其中 $m_1, m_2, \dots, m_n$ 两两互质。

进一步地，当 $m_1, m_2, \dots, m_n$ 两两不一定互质时，求解对应的 x。

复杂度

- Space:
  - Worst Case: $O(n)$
  - Amortized: $O(n)$
  - Best Case: $O(n)$
- Time:
  - Worst Case: $O(n \mathrm{log} m_i)$
  - Amortized: $O(n \mathrm{log} m_i)$
  - Best Case: $O(n)$

算法简述

记 M 为所有 m_i 的最小公倍数，则应有：

$$M = \prod_{i=1}^{n} m_i$$

又记 t_i 为同余方程 $\frac{M}{m_i}t_i \equiv 1 \bmod m_i$ 的最小非负整数解，则存在所求的解 x 满足：

$$x = \sum_{i=1}^{n} a_i \frac{M}{m_i} t_i$$

通解则为 $x^* = x + kM$。

进一步地，推广到当 $m_1, m_2, \dots, m_n$ 不一定两两互质的条件下，我们假定前 $n - 1$ 个同余方程已经被求解，且此时已有 $M = \prod_{i=1}^{n-1} m_i$，$n$ 个同余方程构成的方程组的解应为：

$$t_n M \equiv a_n - x \bmod m_n$$

我们可以调用扩展欧几里得算法求得此同余方程。本质上，扩展中国剩余定理就是求解 n 次扩展欧几里得。

调用方法

- `lli crt_solve(lli[] a, lli[] m, int n)`：求解由 n 个方程构成的同余方程组，其中保证 m 内整数两两互质，返回最小的非负整数解 x。
- `lli excrt_solve(lli[] a, lli[] m, int n)`：求解由 n 个方程构成的同余方程组，其中 m 内整数不一定两两互质，返回最小的非负整数解 x。若该方程组无解，返回 -1。

```
lli crt_solve(lli a[], lli m[], int n)
{
    lli res = 0, lcm = 1, t, tg, x, y;
    rep(i, 1, n)
        lcm *= m[i];
    rep(i, 1, n) {
        t = lcm / m[i];
        exgcd(t, m[i], x, y);
        x = ((x % m[i]) + m[i]) % m[i];
        res = (res + t * x * a[i]) % lcm;
    }
    return (res + lcm) % lcm;
}

lli excrt_solve(lli a[], lli m[], int n)
{
    lli cm = m[1], res = a[1], x, y;
    rep(i, 2, n) {
        lli A = cm, B = m[i], C = (a[i] - res % B + B) % B,
            gcd = exgcd(A, B, x, y),
            Bg = B / gcd;
        if (C % gcd != 0)
            return -1;
        x = (x * (C / gcd)) % Bg;
        res += x * cm;
        cm *= Bg;
```

```
        res = (res % cm + cm) % cm;
    }
    return (res % cm + cm) % cm;
}
```

Tarjan 边双连通分量

- 题目：poj3352
- 依赖：

模板描述

给定 n 个点 m 条边的无向图 $G = (V, E)$，求 G 的边双连通分量。

割边：删掉该边以后无向连通图被分割成两个连通子图，也称作桥。

边双连通图：不存在割边的无向连通图，保证任意两个点间至少存在两条不含共同边的路径。

边双连通分量：一个无向图的极大边双联通子图。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$

算法简述

记数组 dfn_i 为点 i 被 dfs 到的次序编号（时间戳），low_i 为 i 和 i 的子树能够追溯到的最早的堆栈中的节点的时间戳。维护一个堆栈用于储存要处理的连通分量。两遍 dfs，第一次标记所有桥，第二次通过桥的标记求出分量。具体做法见代码。

边双连通图有以下性质：

- 在分量内的任意两个点总可以找到两条边不相同的路径互相到达。
- 若一个边被删除后形成两个边连通图，则该边为原图的桥。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：无向边的数量上限。
- `int[] belong`：点 i 属于第 $belong_i$ 个边双连通分量。

- `int[] bsize`：第 i 个边双连通分量的大小为 $bsize_i$。
- `edge[] bridges`：所有割边的点对，保证 $u < v$。
- `bool edge.is_bridge`：标记该边是否为桥。
- `void add_edge(int u, int v)`：添加一条 $u \leftrightarrow v$ 的无向边。
- `int eval()`：求图 G 的边双连通分量，并返回边双连通分量的个数。
- `void init(int n)`：初始化点数为 n 的图。

```
class Tarjan
{
public:
    struct edge
    {
        int u, v;
        bool is_bridge;
        edge *next, *rev;
    } epool[maxm], *edges[maxn];
    int n, ecnt, dcnt, bcnt;
    int dfn[maxn], low[maxn];
    int belong[maxn], bsize[maxn];
    vector<pair<int, int>> bridges;
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->is_bridge = false;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->is_bridge = false;
        q->next = edges[v]; edges[v] = q;
        p->rev = q; q->rev = p;
        return ;
    }
    void dfs1(int p, int par)
    {
        dfn[p] = low[p] = ++dcnt;
        for (edge *ep = edges[p]; ep; ep = ep->next) {
            int q = ep->v;
            if (!dfn[q]) {
                dfs1(q, p);
                minimize(low[p], low[q]);
                if (low[q] > dfn[p])
                    ep->is_bridge = ep->rev->is_bridge = true;
            } else if (dfn[q] < dfn[p] && q != par) {
                minimize(low[p], dfn[q]);
            }
        }
        return ;
    }
    void dfs2(int p)
    {
        dfn[p] = true;
        belong[p] = bcnt;
        bsize[bcnt] += 1;
```

```
        for (edge *ep = edges[p]; ep; ep = ep->next) {
            if (ep->is_bridge) {
                if (ep->u < ep->v)
                    bridges.push_back(make_pair(ep->u, ep->v));
                continue;
            }
            if (!dfn[ep->v])
                dfs2(ep->v);
        }
        return ;
    }
    int eval(void)
    {
        dcnt = bcnt = 0;
        rep(i, 1, n) {
            dfn[i] = low[i] = 0;
            belong[i] = 0;
        }
        rep(i, 1, n)
            if (!dfn[i])
                dfs1(i, 0);
        rep(i, 1, n)
            dfn[i] = false;  // use dfs[] as vis[]
        bridges.clear();
        rep(i, 1, n)
            if (!dfn[i]) {
                bsize[++bcnt] = 0;
                dfs2(i);
            }
        return bcnt;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        rep(i, 1, n)
            edges[i] = nullptr;
        return ;
    }
} graph;
```

Tarjan 点双连通分量

- 题目：poj1144
- 依赖：

模板描述

给定 n 个点 m 条边的无向图 $G=(V,E)$，求 G 的点双连通分量。

割点：删掉该点以后无向连通图被分割成两个连通子图。

点双连通图：不存在割点的无向连通图。

点双连通分量：一个无向图的极大点双联通子图。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$

算法简述

记数组 dfn_i 为点 i 被 dfs 到的次序编号（时间戳），low_i 为 i 和 i 的子树能够追溯到的最早的堆栈中的节点的时间戳。维护一个堆栈用于储存要处理的连通分量。若回溯时目标节点 low_i 不小于当前点 dfn 值，则不断出栈直到目标节点。具体做法见代码。

点双连通图有以下性质：

- 任意两点间至少存在两条点不重复的路径。
- 图中删去任意一个点都不会改变图的连通性。
- 若点双连通分量之间存在公共点，则这个点为原图的割点。
- 无向连通图中割点必定属于至少两个点双连通分量，非割点属于且恰属于一个。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：无向边的数量上限。
- `int[] belong`：点 i 属于第 $belong_i$ 个点双连通分量。
- `int[] bsize`：第 i 个点双连通分量的大小为 $bsize_i$。
- `bool[] is_cut`：标记点 i 是否为割点。
- `void add_edge(int u, int v)`：添加一条 $u \leftrightarrow v$ 的无向边。
- `int eval()`：求图 G 的点双连通分量，并返回点双连通分量的个数。
- `void init(int n)`：初始化点数为 n 的图。

```
class Tarjan
{
public:
    struct edge
    {
        int u, v;
        edge *next;
```

```
} epool[maxm], *edges[maxn];
int n, ecnt, dcnt, bcnt;
stack<edge*> stk;
int instk[maxn], dfn[maxn], low[maxn];
int belong[maxn], bsize[maxn];
bool is_cut[maxn];
void add_edge(int u, int v)
{
    edge *p = &epool[++ecnt],
         *q = &epool[++ecnt];
    p->u = u; p->v = v;
    p->next = edges[u]; edges[u] = p;
    q->u = v; q->v = u;
    q->next = edges[v]; edges[v] = q;
    return ;
}
void dfs(int p, int par)
{
    int child = 0;
    dfn[p] = low[p] = ++dcnt;
    for (edge *ep = edges[p]; ep; ep = ep->next) {
        int q = ep->v;
        if (!dfn[q]) {
            stk.push(ep);
            child += 1;
            dfs(ep->v, p);
            minimize(low[p], low[q]);
            if (dfn[p] <= low[q]) {
                is_cut[p] = true;
                bcnt += 1;
                edge *eq = nullptr;
                do {
                    eq = stk.top();
                    stk.pop();
                    if (belong[eq->u] != bcnt) {
                        belong[eq->u] = bcnt;
                        bsize[bcnt] += 1;
                    }
                    if (belong[eq->v] != bcnt) {
                        belong[eq->v] = bcnt;
                        bsize[bcnt] += 1;
                    }
                } while (eq->u != p || eq->v != q);
            }
        } else if (dfn[q] < dfn[p] && q != par) {
            stk.push(ep);
            minimize(low[p], dfn[q]);
        }
    }
    if (par == 0 && child == 1)
        is_cut[p] = false;
    return ;
}
int eval(void)
```

```
    {
        while (!stk.empty())
            stk.pop();
        dcnt = bcnt = 0;
        rep(i, 1, n) {
            dfn[i] = low[i] = 0;
            instk[i] = iscut[i] = false;
            belong[i] = 0;
        }
        rep(i, 1, n)
            if (!dfn[i])
                dfs(i, 0);
        return bcnt;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        rep(i, 1, n)
            edges[i] = nullptr;
        return ;
    }
} graph;
```

Dijkstra 最短路

- 题目：hdu2544
- 依赖：

模板描述

给定 n 个点 m 条边的有向图 $G=(V,E)$，$s\in V$ 以及每条边的长度（无负权回路），求点 s 到 V 中任意一点的最短距离及满足距离最短的任意一条最短路径。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O((n+m)\log n)$
  - Amortized: $O((n+m)\log n)$
  - Best Case: $O((n+m)\log n)$

算法简述

松弛操作：对于边 $e(i,j)\in E$，$dist_j := min(dist_j, dist_i + len_e)$

朴素 Dijkstra 算法：从起点开始，松弛每一个可达的点，并依次递归处理这些被松弛的点。

堆优化，将距离源点近的点优先弹出队列，保证最大程度减小无用松弛的次数。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：有向边的数量上限。
- `lli infinit`：规定的无限远距离。
- `lli[] dist`：对于每个点 $i \in [1, n]$，点 i 到源点 s 的距离。
- `edge[] from`：从源点到点 i 的最短路径上指向 i 的边的地址。
- `void add_edge(int u, int v, lli len)`：添加一条 $u \to v$，长度为 len 的有向边。
- `void eval(int s)`：以 s 为源点，计算到 $[1, n]$ 所有点的最短路。
- `int[] get_path(int p)`：获取一个节点编号构成的列表，构成一条 s 到 p 的最短路径。
- `void init(int n)`：初始化点数为 n 的图。

```
class Dijkstra
{
public:
    struct edge
    {
        int u, v;
        lli len;
        edge *next;
    } epool[maxm], *edges[maxn], *from[maxn];
    int n, ecnt;
    lli dist[maxn];
    bool vis[maxn];
    typedef pair<lli, int> pli;
    void add_edge(int u, int v, lli len)
    {
        edge *p = &epool[++ecnt];
        p->u = u; p->v = v; p->len = len;
        p->next = edges[u]; edges[u] = p;
        return ;
    }
    void eval(int s)
    {
        priority_queue<pli, vector<pli>, greater<pli>> pq;
        rep(i, 0, n) {
            vis[i] = false;
            dist[i] = infinit;
            from[i] = nullptr;
        }
        dist[s] = 0;
        pq.push(make_pair(dist[s], s));
```

```
        while (!pq.empty()) {
            pli pr = pq.top();
            pq.pop();
            int p = pr.second;
            if (vis[p])
                continue;
            vis[p] = true;
            for (edge *ep = edges[p]; ep; ep = ep->next)
                if (!vis[ep->v] && dist[p] + ep->len < dist[ep->v])
{
                    dist[ep->v] = dist[p] + ep->len;
                    from[ep->v] = ep;
                    pq.push(make_pair(dist[ep->v], ep->v));
                }
        }
        return ;
    }
    vector<int> get_path(int p)
    {
        stack<int> stk;
        stk.push(p);
        edge *ep = from[p];
        while (ep) {
            stk.push(ep->u);
            ep = from[ep->u];
        }
        vector<int> res;
        while (!stk.empty()) {
            res.push_back(stk.top());
            stk.pop();
        }
        return res;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        rep(i, 1, n)
            edges[i] = nullptr;
        return ;
    }
} graph;
```

Dinic 最大流

- 题目：hdu3549
- 依赖：

模板描述

给定 n 个点 m 条边的有向图 $G = (V, E)$，每条边的权值（即流量），给定源点 s 和汇点 t，求从点 s 到点 t 的最大流量。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O(n^2m)$
  - Amortized: $O(n^2m)$
  - Best Case: $O(n+m)$

算法简述

在最大流的题目中，图被称为网络，而每条边的边权被称作流量。我们可以把最大流问题想象成这样：源点有一个水库有无限吨水，汇点也有一个水库，希望得到最多的水。有一些水管，单位时间内可以输水 $e(i,j)$ 吨。

正式地讲，最大流是指网络中满足弧流量限制条件和平衡条件且具有最大流量的可行流。

定义前向弧为和有向边方向相同的弧，反之称之为后向弧。

定义增广路为一条链，其中链上的前向弧都不饱和，后向弧都非零，从源点开始汇点结束。

增广过程为寻找一条增广路的过程。Dinic 算法的本质就是不断寻找增广路直到找不到为止。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：有向边的数量上限。
- `lli infinit`：规定的无限大流量。
- `void add_edge(int u, int v, lli flow)`：添加一条 $u \to v$，流量为 $flow$ 的有向边。
- `lli eval()`：计算该有向图的最大流。
- `void init(int n, int s, int t)`：初始化点数为 n，源点为 s，汇点为 t 的图。

```
class Dinic
{
public:
    struct edge
    {
        int u, v;
        lli flow;
        edge *next, *rev;
    } epool[maxm], *edges[maxn];
    int n, s, t, ecnt, level[maxn];
    void add_edge(int u, int v, lli flow, lli rflow = 0)
```

```
{
    edge *p = &epool[++ecnt],
         *q = &epool[++ecnt];
    p->u = u; p->v = v; p->flow = flow;
    p->next = edges[u]; edges[u] = p;
    q->u = v; q->v = u; q->flow = rflow;
    q->next = edges[v]; edges[v] = q;
    p->rev = q; q->rev = p;
    return ;
}
bool make_level(void)
{
    rep(i, 1, n)
        level[i] = 0;
    queue<int> que;
    level[s] = 1;
    que.push(s);
    while (!que.empty()) {
        int p = que.front();
        que.pop();
        for (edge *ep = edges[p]; ep; ep = ep->next)
            if (ep->flow > 0 && !level[ep->v]) {
                level[ep->v] = level[p] + 1;
                que.push(ep->v);
            }
        if (level[t] > 0)
            return true;
    }
    return false;
}
lli find(int p, lli mn)
{
    if (p == t)
        return mn;
    lli tmp = 0, sum = 0;
    for (edge *ep = edges[p]; ep && sum < mn; ep = ep->next)
        if (ep->flow && level[ep->v] == level[p] + 1) {
            tmp = find(ep->v, min(mn, ep->flow));
            if (tmp > 0) {
                sum += tmp;
                ep->flow -= tmp;
                ep->rev->flow += tmp;
                return tmp;
            }
        }
    if (sum == 0)
        level[p] = 0;
    return 0;
}
lli eval(void)
{
    lli tmp, sum = 0;
    while (make_level()) {
        bool found = false;
```

```
            while (tmp = find(s, infinit)) {
                sum += tmp;
                found = true;
            }
            if (!found)
                break;
        }
        return sum;
    }
    void init(int n, int s, int t)
    {
        this->n = n;
        this->s = s;
        this->t = t;
        ecnt = 0;
        rep(i, 1, n)
            edges[i] = nullptr;
        return ;
    }
} graph;
```

并查集

- 题目：
- 依赖：

模板描述

给定 n 个集合 $\{1\},\{2\},\ldots,\{n\}$，在线完成以下操作：

- $Query(i,j)$：查询两个数 i,j 是否在一个集合内
- $Merge(i,j)$：合并两个数 i,j 所在集合

复杂度

- Space:
  - Worst Case: $O(n)$
  - Amortized: $O(n)$
  - Best Case: $O(n)$
- $Query$ Time:
  - Worst Case: $O(\log(n))$
  - Amortized: $O(\alpha(n))$
  - Best Case: $O(1)$
- $Merge$ Time:
  - Worst Case: $O(\log(n))$
  - Amortized: $O(\alpha(n))$
  - Best Case: $O(1)$

算法简述

复杂度分析中的 $\alpha(n)$ 函数为反 Ackermann 函数，在一般数据范围内视作常数复杂度。

每个数记一个 $id[i]$ 标记，为该数所在的集合的标志。合并两个集合 p,q 时，仅需设 $id[p]:=q$ 即可。这样形成了一个树结构，但为了避免路径长度变成 $O(n)$ 复杂度，我们需要执行路径压缩。具体地就是在查询某个点 p 的祖先 q 时，将该条路径上所有点的标记 $id[i]$ 修改成 q。这样我们可以保证每个非祖先点被访问的次数与查询的次数之比不超过某个常数。进一步地，为防止不完整的路径压缩，我们将大小更小的集合合并到更大的集合上，即若 $size[p]<size[q]$，则有 $id[p]:=q$，反之 $id[q]:=p$。

在并查集上可维护多种值，例如集合的和、异或和等等。

TODO：各种花式并查集技巧待填坑

调用方法

- `int find(int p)`：查询某点 p 的祖先。
- `bool joined(int p, int q)`：询问两个点 p,q 是否处在一个集合内，即 $Query$ 操作。
- `void join(int p, int q)`：将两个点 p,q 所在集合合并，即 $Merge$ 操作。

```
class DisjointSet
{
public:
    int n, par[maxn], size[maxn];
    lli vsm[maxn];
    int find(int p)
    {
        if (par[p] != p)
            par[p] = find(par[p]);
        return par[p];
    }
    bool joined(int p, int q)
    {
        return find(p) == find(q);
    }
    void join(int p, int q)
    {
        int gp = find(p),
            gq = find(q);
        if (gp == gq)
            return ;
        if (size[gq] < size[gp])
            swap(gq, gp);
        par[gp] = gq;
        size[gq] += size[gp];
        vsm[gq] += vsm[gp];
        return ;
```

```
    }
    void init(int n, lli w[] = nullptr)
    {
        rep(i, 1, n) {
            par[i] = i;
            size[i] = 1;
            vsm[i] = w ? w[i] : 0;
        }
        return ;
    }
} dsu;
```

Euclid 算法

- 题目：
- 依赖：

模板描述

给定整数 a, b，计算 $\gcd(a, b)$。

给定整数 a, b，计算 $\DeclareMathOperator{lcm}{lcm}\lcm(a, b)$。

求解同余方程 $ax + by = \gcd(a, b)$。

复杂度

- Space:
  - Worst Case: $O(1)$
  - Amortized: $O(1)$
  - Best Case: $O(1)$
- Time:
  - Worst Case: $O(\log(a + b))$
  - Amortized: $O(\log(a + b))$
  - Best Case: $O(1)$

算法简述

若有 $r_i = r_{i+1}q_{i+1} + r_{i+2}$，且 $r_n = 0$，则有 $r_{i+2} = r_i \mathrm{mod} r_{i+1}$。由共 $n - 2$ 个式子逆推可知任意的 r_i 可整除 r_n，正确性易得证。

调用方法

- `lli gcd_iterative(lli a, lli b)`：递推式写法的最大公因数，计算 $\gcd(a, b)$。
- `lli gcd(lli a, lli b)`：递归式写法的最大公因数，计算 $\gcd(a, b)$。
- `lli lcm(lli a, lli b)`：计算 a, b 的最小公倍数。
- `lli exgcd(lli a, lli b, lli& x, lli &y)`：求解同余方程 $ax + by = gcd(a, b)$，将方程结果存在 x 和 y 内，调用时传入引用。

```
lli gcd_iterative(lli a, lli b)
{
    if (a < b)
        swap(a, b);
    while (b != 0) {
        lli c = a % b;
        a = b;
        b = c;
    }
    return a;
}

lli gcd(lli a, lli b)
{
    return b == 0 ? a : gcd(b, a % b);
}

lli lcm(lli a, lli b)
{
    return a / gcd(a, b) * b;
}

lli exgcd(lli a, lli b, lli& x, lli& y)
{
    if (b == 0) {
        x = 1, y = 0;
        return a;
    }
    int q = exgcd(b, a % b, y, x);
    y -= lli(a / b) * x;
    return q;
}
```

快速幂

- 题目：bzoj1008, *bzoj3142*
- 依赖：

模板描述

给定整数 a, b，计算 a^b。

复杂度

- Space:
  - Worst Case: $O(1)$
  - Amortized: $O(1)$
  - Best Case: $O(1)$
- Time:
  - Worst Case: $O(\log b)$
  - Amortized: $O(\log b)$

- Best Case: $O(\log b)$

算法简述

$$a^b = \begin{cases} a \cdot a^{b-1} & b \equiv 1 \bmod 2 \\ (a^{\lfloor \frac{b}{2} \rfloor})^2 & b \equiv 0 \bmod 2 \end{cases}$$

用递推方式书写保证运算次数恰为 $\log b$ 轮。

快速幂函数 $fastpow(a,b)$ 可以被拓展成以下形式：$f(a,b,\circ_1,\circ_2)$，其中 $a \circ_1 b \equiv a+b$，$a \circ_2 b = a^b$。一般地，运算符 $\circ_1,\circ_2$ 满足 $\langle \mathbb{Z},\circ_1 \rangle$ 和 $\langle \mathbb{Z},\circ_2 \rangle$ 均为半群，且 $\underbrace{a \circ_1 a \circ_1 \dots \circ_1 a}_{b个a} = a \circ_2 b$。为防止溢出，我们有时使用快速幂的一个变种"快速乘"。注意使用快速乘保证不溢出的快速幂时间复杂度增长到了 $O((\log b)^2)$。

另一种快速乘使用了 `long double` 保证精度，在确保其精度正确性的前提下可代替非模意义下的快速乘。经过粗略检验该方法的出错率应当小于 10^{-8}。

调用方法

- `lli fastmul_old(lli a, lli b, lli m)`：用快速幂写法的 $O(\log b)$ 快速乘，计算 $a \cdot b \bmod m$。
- `lli fastmul(lli a, lli b, lli m)`：利用 `long double` 作为中介运算器的快速乘，计算 $a \cdot b \bmod m$。
- `lli fastpow(lli a, lli b, lli m)`：快速幂，计算 $a^b \bmod m$。
- `lli fastpow_safe(lli a, lli b, lli m)`：适用于模数 $m \geq 2^{31}-1$ 时，为防止溢出，调用了安全的快速乘的快速幂，同样计算 $a^b \bmod m$。

```
lli fastmul_old(lli a, lli b, lli m)
{
    lli res = 0, tmp = a;
    while (b > 0) {
        if (b & 1)
            res = (res + tmp) % m;
        tmp = (tmp + tmp) % m;
        b >>= 1;
    }
    return res;
}

lli fastmul(lli a, lli b, lli m)
{
    a %= m, b %= m;
    return ((a * b - (lli)((long double)a / m * b + 1.0e-8) * m) + m
) % m;
}

lli fastpow(lli a, lli b, lli m)
{
    lli res = 1, tmp = a;
    while (b > 0) {
```

```
        if (b & 1)
            res = res * tmp % m;
        tmp = tmp * tmp % m;
        b >>= 1;
    }
    return res;
}

lli fastpow_safe(lli a, lli b, lli m)
{
    lli res = 1, tmp = a;
    while (b > 0) {
        if (b & 1)
            res = fastmul(res, tmp, m);
        tmp = fastmul(tmp, tmp, m);
        b >>= 1;
    }
    return res;
}
```

Floyd-Warshall 最短路

- 题目：hdu1690
- 依赖：

模板描述

给定 n 个点的有向图 $G = (V, E)$ 以及每条边的长度，求 V 中任意两点之间的最短距离。

复杂度

- Space:
  - Worst Case: $O(n^3)$
  - Amortized: $O(n^3)$
  - Best Case: $O(n^3)$
- Time:
  - Worst Case: $O(n^2)$
  - Amortized: $O(n^2)$
  - Best Case: $O(n^2)$

算法简述

松弛操作：对于边 $e(i,j) \in E$，$dist_{s,j} := min(dist_{s,j}, dist_{s,i} + len_e)$

用边 $(i,k), (k,j)$ 松弛边 (i,j)，循环方式从外到内分别为 k, i, j。

事实上无论何种循环方式，只要执行三次 Floyd 即可保证结果正确。

调用方法

- int maxn：点的数量上限。
- lli infinit：规定的无限远距离。
- lli[][] dist：点 i 到点 j 的最短路径长度。
- void add_edge(int u, int v, lli len)：添加一条 $u \to v$，长度为 len 的有向边。
- void eval()：计算 n 个点中任意两个点间的最短路径长度。
- void init(int n)：初始化点数为 n 的图。

```
class Floyd
{
public:
    lli dist[maxn][maxn];
    int n;
    void add_edge(int u, int v, lli len)
    {
        dist[u][v] = len;
        return ;
    }
    void eval(void)
    {
        rep(k, 1, n)
            rep(i, 1, n)
                rep(j, 1, n)
                    if (dist[i][k] + dist[k][j] < dist[i][j])
                        dist[i][j] = dist[i][k] + dist[k][j];
        return ;
    }
    void init(int n)
    {
        this->n = n;
        rep(i, 1, n)
            rep(j, 1, n)
                dist[i][j] = i == j ? 0 : infinit;
        return ;
    }
} graph;
```

Floyd-Warshall 传递闭包

- 题目：poj3660
- 依赖：

模板描述

给定 n 个点的有向图 $G = (V, E)$，求 V 上任意两点之间的连通性。

复杂度

- Space:

- Worst Case: $O(n^3)$
- Amortized: $O(n^3)$
- Best Case: $O(n^3)$

- Time:
  - Worst Case: $O(n^2)$
  - Amortized: $O(n^2)$
  - Best Case: $O(n^2)$

算法简述

松弛操作：对于边 $e(i,j) \in E$，$conn_{s,j} := conn_{s,j} \vee (conn_{s,i} \wedge conn_{i,j})$

用边 $(i,k),(k,j)$ 松弛边 (i,j)，循环方式从外到内分别为 k,i,j。

调用方法

- `int maxn`：点的数量上限。
- `bool[][] conn`：点 i 到点 j 是否存在路径。
- `void add_edge(int u, int v)`：添加一条 $u \rightarrow v$ 的有向边。
- `void eval()`：计算 n 个点中任意两个点间的连通性。
- `void init(int n)`：初始化点数为 n 的图。

```
class FloydClosure
{
public:
    bool conn[maxn][maxn];
    int n;
    void add_edge(int u, int v)
    {
        conn[u][v] = true;
        return ;
    }
    void eval(void)
    {
        rep(k, 1, n)
            rep(i, 1, n)
                rep(j, 1, n)
                    conn[i][j] |= conn[i][k] && conn[k][j];
        return ;
    }
    void init(int n)
    {
        this->n = n;
        rep(i, 1, n)
            rep(j, 1, n)
                conn[i][j] = false;
        return ;
    }
} graph;
```

HLPP 最大流

- 题目：luogu4722
- 依赖：

模板描述

给定 n 个点 m 条边的有向图 $G = (V, E)$，每条边的权值（即流量），给定源点 s 和汇点 t，求从点 s 到点 t 的最大流量。

复杂度

- Space:
  - Worst Case: $O(n + m)$
  - Amortized: $O(n + m)$
  - Best Case: $O(n + m)$
- Time:
  - Worst Case: $O(n^2\sqrt{m})$
  - Amortized: $O(n^2\sqrt{m})$
  - Best Case: $O(n + m)$

算法简述

HLPP，Highest Label Preflow Push，是预流推进算法的改进版。所谓预流推进就是不停从有剩余流量的点往外推出流量，直到没有任何点可以继续往外推出流量为止。HLPP 是 Tarjan 得到的一个结论，每次推进标号 $level_i$ 最高的点可以保证时间复杂度下降到 $O(n^2\sqrt{m})$。

在极端数据上，HLPP 的速度比普通 ISAP 要快 3 倍以上。用 `vector<edge>` 的写法在事实上比 `edge*` 的链式前向星写法还要快 5 倍。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：有向边的数量上限，使用 `vector` 写法的不需要此常量。
- `lli infinit`：规定的无限大流量。
- `void add_edge(int u, int v, lli flow)`：添加一条 $u \to v$，流量为 $flow$ 的有向边。
- `lli eval()`：计算该有向图的最大流。
- `void init(int n, int s, int t)`：初始化点数为 n，源点为 s，汇点为 t 的图。

```
class HLPPPointer
{
public:
    struct edge
    {
        int u, v;
        lli flow;
```

```
    edge *next, *rev;
} epool[maxm], *edges[maxn];
int n, s, t, ecnt;
int hlevel, level[maxn], cntl[maxn], wcounter;
deque<int> lst[maxn];
vector<int> gap[maxn];
lli dist[maxn];
void add_edge(int u, int v, lli flow, lli rflow = 0)
{
    edge *p = &epool[++ecnt],
         *q = &epool[++ecnt];
    p->u = u; p->v = v; p->flow = flow;
    p->next = edges[u]; edges[u] = p;
    q->u = v; q->v = u; q->flow = rflow;
    q->next = edges[v]; edges[v] = q;
    p->rev = q; q->rev = p;
    return ;
}
void update_height(int p, int nh)
{
    wcounter += 1;
    if (level[p] != n + 2)
        cntl[level[p]] -= 1;
    level[p] = nh;
    if (nh == n + 2)
        return ;
    cntl[nh] += 1;
    hlevel = nh;
    gap[nh].push_back(p);
    if (dist[p] > 0)
        lst[nh].push_back(p);
    return ;
}
void relabel(void)
{
    memclr(cntl, n);
    rep(i, 0, n)
        level[i] = n + 2;
    rep(i, 0, hlevel) {
        lst[i].clear();
        gap[i].clear();
    }
    wcounter = 0;
    queue<int> que;
    level[t] = 0;
    que.push(t);
    while (!que.empty()) {
        int p = que.front();
        que.pop();
        for (edge *ep = edges[p]; ep; ep = ep->next)
            if (level[ep->v] == n + 2 && ep->rev->flow > 0) {
                que.push(ep->v);
                update_height(ep->v, level[p] + 1);
            }
```

```
        hlevel = level[p];
    }
    return ;
}
void push(int p, edge *ep)
{
    if (dist[ep->v] == 0)
        lst[level[ep->v]].push_back(ep->v);
    lli flow = min(dist[p], ep->flow);
    ep->flow -= flow;
    ep->rev->flow += flow;
    dist[p] -= flow;
    dist[ep->v] += flow;
    return ;
}
void discharge(int p)
{
    int nh = n + 2;
    for (edge *ep = edges[p]; ep; ep = ep->next)
        if (ep->flow > 0) {
            if (level[p] == level[ep->v] + 1) {
                push(p, ep);
                if (dist[p] <= 0)
                    return ;
            } else {
                nh = min(nh, level[ep->v] + 1);
            }
        }
    if (cntl[level[p]] > 1) {
        update_height(p, nh);
    } else {
        rep(i, level[p], n + 2 - 1) {
            for (auto j : gap[i])
                update_height(j, n + 2);
            gap[i].clear();
        }
    }
    return ;
}
lli eval(void)
{
    memclr(dist, n);
    dist[s] = infinit;
    dist[t] = - infinit;
    hlevel = 0;
    relabel();
    for (edge *ep = edges[s]; ep; ep = ep->next)
        push(s, ep);
    for (; hlevel >= 0; hlevel--)
        while (!lst[hlevel].empty()) {
            int p = lst[hlevel].back();
            lst[hlevel].pop_back();
            discharge(p);
            if (wcounter > 4 * n)
```

```
                relabel();
            }
        return dist[t] + infinit;
    }
    void init(int n, int s, int t)
    {
        this->n = n;
        this->s = s;
        this->t = t;
        ecnt = 0;
        memclr(edges, n);
        return ;
    }
} graph;

class HLPPVeryFast
{
public:
    struct edge
    {
        int v, rev;
        lli flow;
        edge(int _1, int _2, lli _3) {
            v = _1, rev = _2, flow = _3; }
    };
    vector<edge> edges[maxn];
    int n, s, t, ecnt;
    int hlevel, level[maxn], cntl[maxn], wcounter;
    deque<int> lst[maxn];
    vector<int> gap[maxn];
    lli dist[maxn];
    void add_edge(int u, int v, lli flow, lli rflow = 0)
    {
        edges[u].push_back(edge(v, edges[v].size(), flow));
        edges[v].push_back(edge(u, edges[u].size() - 1, rflow));
        return ;
    }
    void update_height(int p, int nh)
    {
        wcounter += 1;
        if (level[p] != n + 2)
            cntl[level[p]] -= 1;
        level[p] = nh;
        if (nh == n + 2)
            return ;
        cntl[nh] += 1;
        hlevel = nh;
        gap[nh].push_back(p);
        if (dist[p] > 0)
            lst[nh].push_back(p);
        return ;
    }
    void relabel(void)
    {
```

```
        memclr(cntl, n);
        rep(i, 0, n)
            level[i] = n + 2;
        rep(i, 0, hlevel) {
            lst[i].clear();
            gap[i].clear();
        }
        wcounter = 0;
        queue<int> que;
        level[t] = 0;
        que.push(t);
        while (!que.empty()) {
            int p = que.front();
            que.pop();
            for (edge &ep : edges[p])
                if (level[ep.v] == n + 2 && edges[ep.v][ep.rev].flow
> 0) {
                    que.push(ep.v);
                    update_height(ep.v, level[p] + 1);
                }
            hlevel = level[p];
        }
        return ;
    }
    void push(int p, edge& ep)
    {
        if (dist[ep.v] == 0)
            lst[level[ep.v]].push_back(ep.v);
        lli flow = min(dist[p], ep.flow);
        ep.flow -= flow;
        edges[ep.v][ep.rev].flow += flow;
        dist[p] -= flow;
        dist[ep.v] += flow;
        return ;
    }
    void discharge(int p)
    {
        int nh = n + 2;
        for (edge& ep : edges[p])
            if (ep.flow > 0) {
                if (level[p] == level[ep.v] + 1) {
                    push(p, ep);
                    if (dist[p] <= 0)
                        return ;
                } else {
                    nh = min(nh, level[ep.v] + 1);
                }
            }
        if (cntl[level[p]] > 1) {
            update_height(p, nh);
        } else {
            rep(i, level[p], n + 2 - 1) {
                for (auto j : gap[i])
                    update_height(j, n + 2);
```

```
                gap[i].clear();
            }
        }
        return ;
    }
    lli eval(void)
    {
        memclr(dist, n);
        dist[s] = infinit;
        dist[t] = - infinit;
        hlevel = 0;
        relabel();
        for (edge& ep : edges[s])
            push(s, ep);
        for (; hlevel >= 0; hlevel--)
            while (!lst[hlevel].empty()) {
                int p = lst[hlevel].back();
                lst[hlevel].pop_back();
                discharge(p);
                if (wcounter > 4 * n)
                    relabel();
            }
        return dist[t] + infinit;
    }
    void init(int n, int s, int t)
    {
        this->n = n;
        this->s = s;
        this->t = t;
        ecnt = 0;
        rep(i, 1, n)
            edges[i].clear();
        return ;
    }
} graph;
```

匈牙利算法

- 题目：poj3041
- 依赖：

模板描述

给定 n 个点 m 条边的二分图 $G = (V, E)$，求二分图的最大匹配数。

复杂度

- Space:
  - Worst Case: $O(n + m)$
  - Amortized: $O(n + m)$
  - Best Case: $O(n + m)$
- Time:

- Worst Case: $O(nm)$
- Amortized: $O(nm)$
- Best Case: $O(n+m)$

算法简述

二分图 $G=(V,E)$ 的点集 V 可以分割成互补的两个点集 V_1,V_2，且两个点集内分别不存在点集内连边，即 $\forall(u,v)\in E,u\in V_1,u\in V_2$。匈牙利算法试图去匹配任意一个 V_2 内的点，如果该点没有被另一个 V_1 内的点匹配，直接创建点对；反之，记原来已经匹配好的点对 $(u',v')s.t.u'\in V_1,v'\in V_2$，此时可以试图给 u' 再分配一个 V_2 中点 v''。

换言之，匈牙利算法本质上就是询问一个点对是否能找到其他可行匹配，可行则贪心匹配一对即可。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：有向边的数量上限。
- `void add_edge(int u, int v)`：添加一条 $u\to v$ 的有向边。
- `lli eval()`：计算该二分图的匹配数。
- `void init(int n)`：初始化点数为 n 的图。

```
class HungaryMatch
{
public:
    struct edge
    {
        int u, v;
        edge *next;
    };
    edge epool[maxm], *edges[maxn];
    int n, ecnt, from[maxn], vis[maxn];
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt];
        p->u = u; p->v = v;
        p->next = edges[u]; edges[u] = p;
        return ;
    }
    bool find(int p)
    {
        for (edge *ep = edges[p]; ep; ep = ep->next)
            if (!vis[ep->v]) {
                vis[ep->v] = true;
                if (!from[ep->v] || find(from[ep->v])) {
                    from[ep->v] = p;
                    return true;
                }
            }
        return false;
```

```
    }
    int eval(void)
    {
        int res = 0;
        rep(i, 1, n) {
            memclr(vis, n);
            if (find(i))
                res += 1;
        }
        return res;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        memclr(edges, n);
        return ;
    }
} graph;
```

ISAP 最大流

- 题目：hdu3549
- 依赖：

模板描述

给定 n 个点 m 条边的有向图 $G=(V,E)$，每条边的权值（即流量），给定源点 s 和汇点 t，求从点 s 到点 t 的最大流量。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O(n^2m)$
  - Amortized: $O(n^2m)$
  - Best Case: $O(n+m)$

算法简述

ISAP，Improved Shortest Augmenting Path，从汇点开始反向 bfs 建立层次图，但是每次 dfs 的时候每个点的层次随着算法进行不断提高，这样就不用多次 bfs 重建层次图了。当 s 的深度超过 n，则增广完毕，结束算法。

一般地，优化 ISAP 速度能比朴素 Dinic 快 1 倍左右。此外，虽然 HLPP 拥有较优的渐进时间复杂度 $O(n^2\sqrt{m})$，但是常数较大，在普通随机数据上表现也不如 ISAP。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：有向边的数量上限。
- `lli infinit`：规定的无限大流量。
- `void add_edge(int u, int v, lli flow)`：添加一条 $u \to v$，流量为 $flow$ 的有向边。
- `lli eval()`：计算该有向图的最大流。
- `void init(int n, int s, int t)`：初始化点数为 n，源点为 s，汇点为 t 的图。

```
class ISAP
{
public:
    struct edge
    {
        int u, v;
        lli flow;
        edge *next, *rev;
    } epool[maxm], *edges[maxn], *cedge[maxn];
    int n, s, t, ecnt, level[maxn], gap[maxn];
    lli maxflow;
    void add_edge(int u, int v, lli flow, lli rflow = 0)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->flow = flow;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->flow = rflow;
        q->next = edges[v]; edges[v] = q;
        p->rev = q; q->rev = p;
        return ;
    }
    void make_level(void)
    {
        memclr(level, n);
        memclr(gap, n);
        queue<int> que;
        level[t] = 1;
        gap[level[t]] += 1;
        que.push(t);
        while (!que.empty()) {
            int p = que.front();
            que.pop();
            for (edge *ep = edges[p]; ep; ep = ep->next)
                if (!level[ep->v]) {
                    level[ep->v] = level[p] + 1;
                    gap[level[ep->v]] += 1;
                    que.push(ep->v);
                }
        }
        return ;
```

```
    }
    lli find(int p, lli flow)
    {
        if (p == t)
            return flow;
        lli used = 0;
        for (edge *ep = cedge[p]; ep; ep = ep->next)
            if (ep->flow > 0 && level[ep->v] + 1 == level[p]) {
                lli tmp = find(ep->v, min(ep->flow, flow - used));
                if (tmp > 0) {
                    ep->flow -= tmp;
                    ep->rev->flow += tmp;
                    used += tmp;
                    cedge[p] = ep;
                }
                if (used == flow)
                    return used;
            }
        gap[level[p]] -= 1;
        if (gap[level[p]] == 0)
            level[s] = n + 1;
        level[p] += 1;
        gap[level[p]] += 1;
        cedge[p] = edges[p];
        return used;
    }
    lli eval(void)
    {
        lli res = 0;
        make_level();
        rep(i, 1, n)
            cedge[i] = edges[i];
        while (level[s] <= n)
            res += find(s, infinit);
        return res;
    }
    void init(int n, int s, int t)
    {
        this->n = n;
        this->s = s;
        this->t = t;
        ecnt = 0;
        memclr(edges, n);
        return ;
    }
} graph;
```

Knuth-Morris-Pratt 字符串匹配

- 题目：hdu1686, hdu2087
- 依赖：

模板描述

给定模板串 str 和模式串 $patt$，求 str 有多少个（相交或互不相交）子串与 $patt$ 相等。

记 $|str| = n, |patt| = m$。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$

算法简述

记 $next_i$ 为前 i 个字符中最长的长度 k，使得 $patt[1,k] = patt[i-k+1,n]$。由于 $next$ 数组可以在 $O(m)$ 时间内用递推法求出，且可以辅助跳过已经匹配过的后缀（前缀符合，所以下一个符合匹配条件的就直接跳到 $next_j$ 即可。

本模板内数组下标从 0 开始。

调用方法

- `int maxn`：字符串长度上限。
- `int[] match(bool overlap)`：用模式串 $patt$ 匹配模板串 str，参数 $overlap$ 决定匹配时是否允许匹配结果相互交叉。
- `void init_patt(int m, char[] patt)`：加载模式串。
- `void init_str(int n, char[] str)`：加载模板串。

```
class KMP
{
public:
    int m, patt[maxn], nxt[maxn];
    int n, str[maxn];
    void get_next(void)
    {
        int i = 0, j = -1;
        nxt[0] = -1;
        for (int i = 0, j = -1; i < m; ) {
            if (j == -1 || patt[i] == patt[j])
                i += 1, j += 1, nxt[i] = j;
            else
                j = nxt[j];
        }
        return ;
    }
```

```
    vector<int> match(bool overlap = true)
    {
        vector<int> res;
        for (int i = 0, j = 0; i < n; ) {
            if (j == -1 || str[i] == patt[j])
                i += 1, j += 1;
            else
                j = nxt[j];
            if (j == m)
                res.push_back(i), j = overlap ? nxt[j] : 0;
        }
        return res;
    }
    void init_patt(int m, char patt[])
    {
        this->m = m;
        rep(i, 0, m - 1)
            this->patt[i] = patt[i];
        memclr(nxt, m);
        get_next();
        return ;
    }
    void init_str(int n, char str[])
    {
        this->n = n;
        rep(i, 0, n - 1)
            this->str[i] = str[i];
        return ;
    }
} kmp;
```

Kruskal 最小生成树

- 题目：hdu3371
- 依赖：disjoint_set

模板描述

给定 n 个点 m 条边的无向图 $G = (V, E)$，求图 G 的最小生成树 $G' = (V, E' \subseteq E)$，且 $\sum_{e \in E'} |e|$ 最小。

复杂度

- Space:
  - Worst Case: $O(m)$
  - Amortized: $O(m)$
  - Best Case: $O(m)$
- Time:
  - Worst Case: $O(mlogm)$
  - Amortized: $O(mlogm)$
  - Best Case: $O(nlogm)$

算法简述

我们可以贪心地选择最小的边，将它们连起来，如果它们本来不在一个连通块上的话。理由如下：倘若存在边 $e_1 = (u, v), e_2 = (v, w), e_3 = (u, w)$，且 $|e_1| < |e_2| < |e_3|$，则同样可以连起来的方法中总是以 e_1 和 e_2 优先。若我们舍弃了一个较短的边，则在达成条件的前提下总是比选择短边的方案要差。

如此我们用一个 $O(m\log m)$ 的排序来贪心，用并查集维护点联通，若存在事先已连接好的点则将其在并查集上预处理，即可。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：无向边的数量上限。
- `void add_edge(int u, int v, lli len)`：添加一条 $u \leftrightarrow v$，长度为 len 的无向边。
- `void join(int u, int v)`：预先连接点 u 和点 v。
- `lli eval()`：计算图 G 的最小生成树的边权之和，在代码中对应位置修改可求得该生成树的构造。
- `void init(int n)`：初始化点数为 n 的图。

```
class Kruskal
{
public:
    struct edge
    {
        int u, v;
        lli len;
        bool operator < (const edge& b) const
        {
            return this->len < b.len;
        }
    } edges[maxm];
    int n, m, mst_cnt;
    void add_edge(int u, int v, lli len)
    {
        edge *ep = &edges[++m];
        ep->u = u; ep->v = v; ep->len = len;
        return ;
    }
    void join(int u, int v)
    {
        if (!dsu.joined(u, v)) {
            dsu.join(u, v);
            mst_cnt += 1;
        }
        return ;
    }
    lli eval(void)
    {
        lli min_span = 0;
```

```
        sort(edges + 1, edges + m + 1);
        rep(i, 1, m) {
            if (mst_cnt >= n - 1)
                break;
            int u = edges[i].u, v = edges[i].v, len = edges[i].len;
            if (dsu.joined(u, v))
                continue;
            dsu.join(u, v);
            // printf("add_edge %d -> %d : %lld\n", u, v, len);
            min_span += len;
            mst_cnt += 1;
        }
        if (mst_cnt < n - 1)
            min_span = -1;
        return min_span;
    }
    void init(int n)
    {
        this->n = n;
        m = 0;
        mst_cnt = 0;
        dsu.init(n);
        return ;
    }
} graph;
```

线性筛

- 题目：
- 依赖：

模板描述

计算 $[1, n]$ 内的所有正整数的 $\varphi(i)$，以及筛出该区间内的质数。

复杂度

- Space:
  - Worst Case: $O(n)$
  - Amortized: $O(n)$
  - Best Case: $O(n)$
- Time:
  - Worst Case: $O(n)$
  - Amortized: $O(n)$
  - Best Case: $O(n)$

算法简述

在朴素的 $O(n^2)$ 筛法上优化成为了 $O(n)$ 的欧拉筛，使每个数不会被筛两次，且保证所有合数被消除。

利用 $\varphi(x)$ 的积性函数性质可以在筛质数的过程中计算 φ 函数。

调用方法

- `int maxn`：筛选的质数的数量上限。
- `int maxp`：$[1,n]$ 内质数的数量上限。
- `void filter(int n)`：筛选出 $[1,n]$ 范围内的质数及其 $\varphi(i)$ 值。
- `bool[] isprime`：长度为 $maxn$ 的数组，代表该位置是否为质数。
- `int[] primes`：长度为 $maxp$ 的数组，$primes[0]$ 代表一共有多少质数，$primes[i]$ 为该范围内第 i 个质数。$[1,n]$ 内质数的个数满足：

$$primes[0] = \begin{cases} 1229 & n = 10^4 \\ 9592 & n = 10^5 \\ 78498 & n = 10^6 \\ 664579 & n = 10^7 \\ 5761455 & n = 10^8 \end{cases}$$

- `int[] phi`：长度为 $maxn$ 的数组，满足 $phi[i] = \varphi(i)$。

```
bool isprime[maxn];
int primes[maxp];
int phi[maxn];

void filter(int n)
{
    rep(i, 2, n)
        isprime[i] = true;
    isprime[1] = false;
    primes[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (isprime[i]) {
            primes[++primes[0]] = i;
            phi[i] = i - 1;
        }
        for (int j = 1; j <= primes[0] && i * primes[j] <= n; j++) {
            isprime[i * primes[j]] = false;
            if (i % primes[j] == 0) {
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }
    return ;
}
```

矩阵

- 题目：
- 依赖：

模板描述

给定矩阵 A, B，对矩阵进行运算。

复杂度

- Space:
  - Worst Case: $O(n^2)$
  - Amortized: $O(n^2)$
  - Best Case: $O(n^2)$
- Time:
  - Worst Case: $O(n^2)$
  - Amortized: $O(n^2)$
  - Best Case: $O(n^2)$
- Multiplication Time:
  - Worst Case: $O(n^3)$
  - Amortized: $O(n^3)$
  - Best Case: $O(n^3)$
- Determinant Time:
  - Worst Case: $O(n^3)$
  - Amortized: $O(n^3)$
  - Best Case: $O(n^3)$
- Inversion Time:
  - Worst Case: $O(n^5)$
  - Amortized: $O(n^5)$
  - Best Case: $O(n^5)$

算法简述

矩阵求行列式值方法：逐行消掉元素，仅保留上三角矩阵，记录扩倍的值，最后将主对角线上元素相乘除掉消掉元素过程中乘上的值。

矩阵求行列式的定义法：$|A| = \sum_{p_1,p_2,\dots,p_n} (-1)^{a(p_1,p_2,\dots,p_n)} \cdot a_{1,p_1} \cdot a_{2,p_2} \cdots a_{n,p_n}$，递归求或递推求均可。

矩阵求逆的方法：$A^{-1} = \frac{A^*}{|A|}$，其中 A^* 为 A 的增广矩阵，$A^*_{j,i}$ 为 A 去除第 i 行第 j 列对应的代数余子式。

调用方法

- `typedef typ`：矩阵内元素的类型。
- `int maxn`：矩阵的长宽上限。

- `typ operator () (int i, int j)`：寻址第 i 行第 j 列。
- `matrix eye(int n)`：生成 $n\times n$ 的单位矩阵。
- `matrix zeros(int n, int m)`：生成 $n\times m$ 的零矩阵。
- `matrix ones(int n, int m)`：生成 $n\times m$ 的各位均为 1 的矩阵。
- `matrix A + matrix B = matrix`：求 $A+B$。
- `matrix A - matrix B = matrix`：求 $A-B$。
- `matrix A += matrix B`：求值 $A:=A+B$。
- `matrix A -= matrix B`：求值 $A:=A-B$。
- `matrix prod(matrix A, matrix B)`：若矩阵 B 维度与 A 相同，则求两个矩阵对应位置相乘得到的新矩阵 $A\ .^*\ B$；若 B 为列向量，则 A 的每一列与 B 对应元素两两相乘；若 B 为行向量，则 A 的每一行与 B 对应元素两两相乘；若 B 为 1×1 矩阵，则 A 每个元素均与 $B_{1,1}$ 相乘。
- `matrix prod(matrix A, typ b)`：求矩阵 A 和常数 b 的积 $b\cdot A$。
- `matrix A * matrix B = matrix`：求两个矩阵相乘 $A\times B$。
- `matrix A * typ b = matrix`：求矩阵 A 和常数 b 的积 $b\cdot A$。
- `typ a * matrix B = matrix`：求常数 a 和矩阵 $$B 的积 $a\cdot B$。
- `matrix A *= matrix B`：求值 $A:=A\times B$。
- `matrix A *= typ b`：求值 $A:=b\cdot A$。
- `matrix pow(matrix A, lli b)`：求 A^b。
- `matrix transpose(matrix A)`：求矩阵 A 的转置 A^T。
- `matrix det_def(matrix A)`：用定义法（$O(n^{n+1})$）求矩阵 A 的行列式 $|A|$ 。
- `matrix det(matrix A)`：用高斯消元法求矩阵 A 的行列式 $|A|$。
- `matrix inv(matrix A)`：求矩阵 A 的逆。

```
typedef llf typ;

struct matrix
{
protected:
    int n, m;
    typ arr[maxn][maxn];
public:
    matrix(void)
    {
        n = m = 1;
        arr[0][0] = 0;
    }
    matrix(int n, int m)
    {
        this->n = n, this->m = m;
        rep(i, 0, n - 1)
            rep(j, 0, m - 1)
                arr[i][j] = 0;
    }
    typ& operator () (int i, int j)
    {
```

```
        if (i < 1 || i > n || j < 1 || j > m)
            throw std::out_of_range("invalid indices");
        return arr[i - 1][j - 1];
    }
    matrix operator () (int top, int bottom, int left, int right);
    friend matrix eye(int);
    friend matrix zeros(int, int);
    friend matrix ones(int, int);
    friend matrix operator + (const matrix&, const matrix&);
    friend matrix operator - (const matrix&, const matrix&);
    matrix& operator += (const matrix&);
    matrix& operator -= (const matrix&);
    friend matrix prod(const matrix&, const matrix&);
    friend matrix prod(const matrix&, const typ);
    friend matrix operator * (const matrix&, const matrix&);
    friend matrix operator * (const matrix&, const typ);
    friend matrix operator * (const typ, const matrix&);
    matrix& operator *= (const matrix&);
    matrix& operator *= (const typ);
    friend matrix pow(const matrix& a, lli b);
    friend matrix transpose(const matrix&);
    friend typ det_def(const matrix&);
    friend typ det(const matrix&);
    friend matrix inv(const matrix&);
    friend ostream& operator << (ostream&, const matrix&);
};

matrix matrix::operator () (int top, int bottom, int left, int right
)
{
    if (top < 1 || top > n || bottom < 1 || bottom > n || top > bott
om ||
        left < 1 || left > m || right < 1 || right > m || left > rig
ht)
        throw std::out_of_range("invalid indices");
    int h = bottom - top + 1, w = right - left + 1;
    matrix mat(h, w);
    rep(i, 0, h - 1)
        rep(j, 0, w - 1)
            mat.arr[i][j] = arr[top - 1 + i][left - 1 + j];
    return std::move(mat);
}

matrix eye(int n)
{
    matrix mat(n, n);
    rep(i, 0, n - 1)
        mat.arr[i][i] = 1;
    return std::move(mat);
}

matrix zeros(int n, int m = -1)
{
```

```
    if (m == -1)
        m = n;
    matrix mat(n, m);
    return std::move(mat);
}

matrix ones(int n, int m = -1)
{
    if (m == -1)
        m = n;
    matrix mat(n, m);
    rep(i, 0, n - 1)
        rep(j, 0, m - 1)
            mat.arr[i][j] = 1;
    return std::move(mat);
}

matrix operator + (const matrix& a, const matrix& b)
{
    matrix c(a.n, a.m);
    if (b.n != c.n || b.m != c.m)
        throw std::domain_error("nonconformant arguments");
    rep(i, 0, c.n - 1)
        rep(j, 0, c.m - 1)
            c.arr[i][j] = a.arr[i][j] + b.arr[i][j];
    return std::move(c);
}

matrix operator - (const matrix& a, const matrix& b)
{
    matrix c(a.n, a.m);
    if (b.n != c.n || b.m != c.m)
        throw std::domain_error("nonconformant arguments");
    rep(i, 0, c.n - 1)
        rep(j, 0, c.m - 1)
            c.arr[i][j] = a.arr[i][j] - b.arr[i][j];
    return std::move(c);
}

matrix& matrix::operator += (const matrix& b)
{
    if (b.n != n || b.m != m)
        throw std::domain_error("nonconformant arguments");
    rep(i, 0, n - 1)
        rep(j, 0, m - 1)
            arr[i][j] = arr[i][j] + b.arr[i][j];
    return *this;
}

matrix& matrix::operator -= (const matrix& b)
{
    if (b.n != n || b.m != m)
        throw std::domain_error("nonconformant arguments");
```

```
    rep(i, 0, n - 1)
        rep(j, 0, m - 1)
            arr[i][j] = arr[i][j] - b.arr[i][j];
    return *this;
}

matrix prod(const matrix& a, const matrix& b)
{
    matrix c(a.n, a.m);
    if (a.n == b.n && a.m == b.m) {
        rep(i, 0, a.n - 1)
            rep(j, 0, a.m - 1)
                c.arr[i][j] = a.arr[i][j] * b.arr[i][j];
    } else if (a.n == b.n && b.m == 1) {
        rep(i, 0, a.n - 1)
            rep(j, 0, a.m - 1)
                c.arr[i][j] = a.arr[i][j] * b.arr[i][0];
    } else if (a.m == b.m && b.n == 1) {
        rep(i, 0, a.n - 1)
            rep(j, 0, a.m - 1)
                c.arr[i][j] = a.arr[i][j] * b.arr[0][j];
    } else if (b.n == 1 && b.m == 1) {
        rep(i, 0, a.n - 1)
            rep(j, 0, a.m - 1)
                c.arr[i][j] = a.arr[i][j] * b.arr[0][0];
    } else {
        throw std::domain_error("nonconformant arguments");
    }
    return std::move(c);
}

matrix prod(const matrix& a, typ b)
{
    matrix c(a.n, a.m);
    rep(i, 0, a.n - 1)
        rep(j, 0, a.m - 1)
            c.arr[i][j] = a.arr[i][j] * b;
    return std::move(c);
}

matrix operator * (const matrix& a, const matrix& b)
{
    if (a.m != b.n)
        throw std::domain_error("nonconformant arguments");
    matrix c(a.n, b.m);
    rep(i, 0, c.n - 1)
        rep(j, 0, c.m - 1) {
            c.arr[i][j] = 0;
            rep(k, 0, a.m - 1)
                c.arr[i][j] += a.arr[i][k] * b.arr[k][j];
        }
    return std::move(c);
}
```

```
matrix operator * (const matrix& a, const typ b)
{
    return std::move(prod(a, b));
}

matrix operator * (const typ a, const matrix& b)
{
    return std::move(prod(b, a));
}

matrix& matrix::operator *= (const matrix& b)
{
    if (m != b.n)
        throw std::domain_error("nonconformant arguments");
    *this = *this * b;
    return *this;
}

matrix& matrix::operator *= (const typ b)
{
    rep(i, 0, n - 1)
        rep(j, 0, m - 1)
            arr[i][j] = arr[i][j] * b;
    return *this;
}

matrix pow(const matrix& a, lli b)
{
    if (a.n != a.m)
        throw std::domain_error("nonconformant arguments");
    matrix res = eye(a.n), tmp = a;
    while (b > 0) {
        if (b & 1)
            res *= tmp;
        tmp *= tmp;
        b >>= 1;
    }
    return std::move(res);
}

matrix transpose(const matrix& a)
{
    matrix mat(a.m, a.n);
    rep(i, 0, a.n - 1)
        rep(j, 0, a.m - 1)
            mat.arr[j][i] = a.arr[i][j];
    return std::move(mat);
}

typ det_def(const matrix& a)
{
    if (a.n != a.m)
```

```
        throw std::domain_error("nonconformant arguments");
    int n = a.n;
    typ res = 0;
    static int vec[maxn];
    static bool vis[maxn];
    rep(i, 0, n - 1)
        vec[i] = vis[i] = 0;
    stack<pair<int, int>> dfs_stk;  // dfs stack
    rep_(i, n - 1, 0)
        dfs_stk.push(make_pair(1, i + 1));
    while (!dfs_stk.empty()) {
        pair<int, int> pr = dfs_stk.top();
        dfs_stk.pop();
        int depth = pr.first,
            p = abs(pr.second) - 1,
            rm = pr.second < 0;
        if (rm) {  // restore previous state
            vis[p] = false;
            vec[depth - 1] = 0;
            continue;
        } else if (depth == n) {  // final state
            vis[p] = true;
            vec[depth - 1] = p + 1;
            // calculate count inverse
            lli cnt_inv = 0;
            rep(i, 0, n - 1)
                rep(j, i + 1, n - 1)
                    if (vec[i] > vec[j])
                        cnt_inv += 1;
            // (-1) ^ cnt_inv(vec) * sum(arr[i][vec[i]])
            typ tmp = 0;
            tmp = cnt_inv % 2 == 1 ? -1 : 1;
            rep(i, 0, n - 1)
                tmp *= a.arr[i][vec[i] - 1];
            // sum(tmp)
            res += tmp;
            // restore previous state
            vis[p] = false;
            vec[depth - 1] = 0;
            continue;
        }
        // mark restoration state
        dfs_stk.push(make_pair(depth, - (p + 1)));
        vis[p] = true;
        vec[depth - 1] = p + 1;
        // children states
        rep_(i, n - 1, 0)
            if (!vis[i])
                dfs_stk.push(make_pair(depth + 1, i + 1));
    }
    return res;
}

typ det(const matrix& a)
```

```
{
    if (a.n != a.m)
        throw std::domain_error("nonconformant arguments");
    int n = a.n;
    matrix m(n, n);
    rep(i, 0, n - 1)
        rep(j, 0, n - 1)
            m.arr[i][j] = a.arr[i][j];
    // arrange rows
    typ comp = 1;  // compensate
    rep_(i, n - 1, 0) {
        // find non-zero row to dissipate other values
        int flag = -1;
        rep_(j, i, 0)
            if (m.arr[i][j] != 0) {
                flag = j;
                break;
            }
        if (flag == -1)
            return 0;
        // swap rows
        if (flag != i)
            rep(j, 0, n - 1)
                swap(m.arr[j][flag], m.arr[j][i]);
        // eliminate other rows
        rep(j, 0, i - 1) {
            typ pa = m.arr[i][j],
                pb = m.arr[i][i];
            comp *= pb;
            rep(k, 0, n - 1)
                m.arr[k][j] = m.arr[k][j] * pb - m.arr[k][i] * pa;
        }
    }
    // multiply diagonal elements
    typ res = 1;
    rep(i, 0, n - 1)
        res *= m.arr[i][i];
    // divide by compensate
    res /= comp;
    return res;
}

matrix inv(const matrix& a)
{
    if (a.n != a.m)
        throw std::domain_error("nonconformant arguments");
    int n = a.n;
    matrix b(n, n), tmp(n - 1, n - 1);
    typ det_a = det(a);
    if (det_a == 0)
        throw std::invalid_argument("not inversible");
    rep(i, 0, n - 1)
        rep(j, 0, n - 1) {
            rep(_i, 0, n - 2)
```

```
                rep(_j, 0, n - 2)
                    tmp.arr[_i][_j] = a.arr[_i < i ? _i : (_i + 1)]
                                        [_j < j ? _j : (_j + 1)];
            b.arr[j][i] = ((i + j) % 2 == 1 ? -1 : 1) * det(tmp) / d
et_a;
        }
    return std::move(b);
}

ostream& operator << (ostream& out, const matrix& mat)
{
    rep(i, 0, mat.n - 1) {
        out << "|";
        rep(j, 0, mat.m - 1)
            out << " " << mat.arr[i][j];
        out << " |";
        if (i < mat.n - 1)
            out << "\n";
    }
    return out;
}
```

Miller-Rabin 质数判别法

- 题目：
- 依赖：fast_exponentiation

模板描述

给定质数 n，判定 n 是否为质数。

复杂度

- Space:
  - Worst Case: $O(k)$
  - Amortized: $O(k)$
  - Best Case: $O(k)$
- Time:
  - Worst Case: $O(k\log^2 n)$
  - Amortized: $O(k\log^2 n)$
  - Best Case: $O(k\log^2 n)$

算法简述

Fermat 小定理：若 p 为质数，则必有 $a^{p-1} \equiv 1\bmod p$。

反之，若有 $a^{p-1} \equiv 1\bmod p$，则 p 大概率是质数，若 p 为合数定义为 Carmichael 数。

二次探测：如果 p 是一个质数，且 $0 < x < p$，则方程 $x^2 \equiv 1 \bmod p$ 的解为 $x = 1$ 或 $x = p - 1$。

采用多个质数多次对某数 n 进行上述检测，若次数足够多可以确定 n 是否为质数。经过古今中外无数勇士的贡献与检验，得到一组最少的质数表如下：

$$p_i = \begin{cases} 2 & n \le 2.04 \cdot 10^3 \\ 31,73 & n \le 9.08 \cdot 10^6 \\ 2,7,61 & n \le 4.75 \cdot 10^9 \\ 2,13,23,1662803 & n \le 1.12 \cdot 10^{12} \\ 2,3,5,7,11,13,17,19,23,29,31,37 & n \le 3.18 \cdot 10^{23} \end{cases}$$

在 `is_prime.samples` 可修改该质数表，其中 `samples[0]` 代表质数表的大小（上述复杂度分析中的常数 k 也指代质数表的大小。需要注意的是，由于复杂度太高，Miller-Rabin 算法不应用于筛选质数。

调用方法

- `bool miller_rabin_test(lli n, lli p)`：利用质数 p 以一定概率检验 n 是否是质数。
- `bool is_prime(lli n)`：利用事先确定的质数表确定地检验 n 是否为质数。

```
bool miller_rabin_test(lli n, lli k)
{
    if (fastpow(k, n - 1, n) != 1)
        return false;
    lli t = n - 1, tmp;
    while (t % 2 == 0) {
        t >>= 1;
        tmp = fastpow(k, t, n);
        if (tmp != 1 && tmp != n - 1)
            return false;
        if (tmp == n - 1)
            return true;
    }
    return true;
}

bool is_prime(lli n)
{
    if (n == 1 || (n > 2 && n % 2 == 0))
        return false;
    // lli samples[13] = { 12, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 3
1, 37 };
    lli samples[4] = { 4, 2, 7, 61 };
    rep(i, 1, samples[0]) {
        if (n == samples[i])
            return true;
        if (n > samples[i] && !miller_rabin_test(n, samples[i]))
            return false;
    }
```

```
    return true;
}
```

朴素质因数分解

- 题目：
- 依赖：linear_sieve

模板描述

给定整数 n，求 n 的所有质因子。

复杂度

- Space:
  - Worst Case: $O(1)$
  - Amortized: $O(1)$
  - Best Case: $O(1)$
- Time:
  - Worst Case: $O(\frac{\sqrt{n}}{\log\sqrt{n}})$
  - Amortized: $O(\frac{\sqrt{n}}{\log\sqrt{n}})$
  - Best Case: $O(\frac{\sqrt{n}}{\log\sqrt{n}})$

算法简述

整数 n 最多含有一个大于 $\sqrt{n}$ 的质因子，所以仅需暴力判别小于等于 $\sqrt{n}$ 的所有质数即可。

调用该方法前需先用线性筛求出一定范围内的所有质数。

调用方法

- `lli[] factorize(lli n)`：质因数分解 n，将结果（可能重复地）按顺序放入结果中。例如若 $n = 2^3 \cdot 3$，则 $factors = [2,2,2,3]$。

```
vector<lli> factorize(lli n)
{
    lli tmp = n;
    vector<lli> factors;
    rep(i, 1, primes[0]) {
        lli p = primes[i];
        if (p * p > n || tmp <= 1)
            break;
        while (tmp % p == 0) {
            factors.push_back(p);
            tmp /= p;
        }
    }
    if (tmp > 1)
```

```
        factors.push_back(tmp);
    return factors;
}
```

Pollard's Rho 质因数分解

- 题目：
- 依赖：euclid_gcd, fast_exponentiation

模板描述

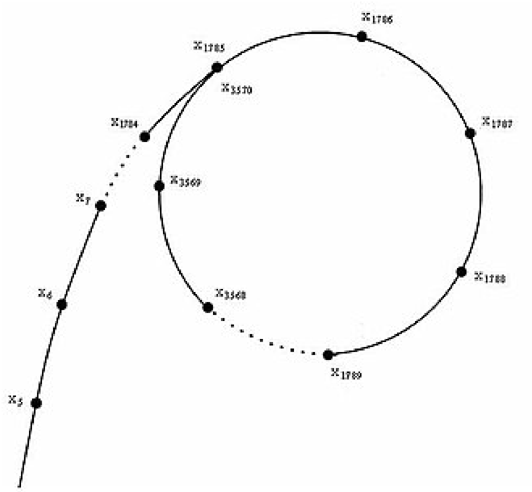
给定整数 n，求 n 的所有质因子。

复杂度

- Space:
  - Worst Case: $O(1)$
  - Amortized: $O(1)$
  - Best Case: $O(1)$
- Time:
  - Worst Case: $O(n)$
  - Amortized: $O(n^{\frac{1}{4}})$
  - Best Case: $O(\sqrt{p})$

算法简述

构造伪随机生成器 $g(x)$，则生成的序列为 x_0，$x_1 = g(x_0)$，$x_2 = g(g(x_0))$，…，$x_n = g(x_{n-1})$。这时候我们发现当 $g(x)$ 的值域有限时它一定会重复，并且最终会成环（这也是为什么算法被称为 ρ 的原因），如下图：



Rho

根据生日悖论，如果共存在 m 种可能的值，则在成环之前出现的值的数量期望为 $\sqrt{m}$ 个。随后我们只需用 Floyd 判环算法找到这个环即可。因此算法的期望

复杂度为 $O(\sqrt{m}) = O(n^{\frac{1}{4}})$，或者若最小因子为 p，其期望复杂度也可记为 $O(\sqrt{p})$。

事实上这个算法在分解大量数字的情况下性能不如借用线性筛的朴素算法好。

调用方法

- `void pollard_rho(lli n, lli[] factors)`：随机化质因数分解 n，将结果无序地放入 $factors$ 中。
- `lli[] pollard_rho(lli n)`：随机化质因数分解 n，有序地返回所有质因子。例如若 $n = 2^3 \cdot 3$，则 $factors = [2,2,2,3]$。

```
void pollard_rho(lli n, vector<lli>& factors)
{
    if (is_prime(n)) {
        factors.push_back(n);
        return ;
    }
    lli a, b, c, d;
    while (true) {
        c = rand() % n;
        a = b = rand() % n;
        b = (fastmul(b, b, n) + c) % n;
        while (a != b) {
            d = a - b;
            d = gcd(abs(d), n);
            if (d > 1 && d < n) {
                pollard_rho(d, factors);
                pollard_rho(n / d, factors);
                return ;
            }
            a = (fastmul(a, a, n) + c) % n;
            b = (fastmul(b, b, n) + c) % n;
            b = (fastmul(b, b, n) + c) % n;
        }
    }
    return ;
}

vector<lli> pollard_rho(lli n)
{
    lli tmp = n;
    vector<lli> factors;
    pollard_rho(n, factors);
    sort(factors.begin(), factors.end());
    return factors;
}
```

Prim 最小生成树

- 题目：hdu1102

- 依赖：

模板描述

给定 n 个点 m 条边的无向图 $G=(V,E)$，求图 G 的最小生成树 $G'=(V,E'\subseteq E)$，且 $\sum_{e\in E'}|e|$ 最小。

复杂度

- Space:
  - Worst Case: $O(n^2)$
  - Amortized: $O(n^2)$
  - Best Case: $O(n^2)$
- Time:
  - Worst Case: $O(n^2)$
  - Amortized: $O(n^2)$
  - Best Case: $O(n^2)$

算法简述

以任意点为起点，维护一个点集 S，初始为 $S=\{1\}$。选择一个点 $p\notin S$ 使得在所有的满足 $a\in S,b\notin S$ 的点对中 p 对应到某一个 $dist_{a,b}$ 最小的点对。随后将 p 加入 S。显然这样的做法是对的，证法类同 Kruskal 算法。

注意 Fibonacci 堆的常数较大，所以邻接表写法的 Fibonacci 堆优化 Prim 的复杂度虽然是 $O(m+n\log n)$ 的，其运行速度将大大不如 $O((n+m)\log n)$ 的二叉堆优化 Prim。进一步地，堆优化 Prim 在稠密图上表现并不明显好于朴素的 Prim，且常数较大。故若非稠密图，尽量应采用 Kruskal 算法。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：无向边的数量上限。
- `lli infinit`：规定的无限远距离。
- `void add_edge(int u, int v, lli len)`：添加一条 $u\leftrightarrow v$，长度为 len 的无向边。
- `void join(int u, int v)`：预先连接点 u 和点 v。
- `lli eval()`：计算图 G 的最小生成树的边权之和，在代码中对应位置修改可求得该生成树的构造。
- `void init(int n)`：初始化点数为 n 的图。

```
class Prim
{
public:
    lli dist[maxn][maxn];
    int n, vis[maxn], min_cost[maxn];
    void add_edge(int u, int v, lli len)
    {
```

```
        dist[u][v] = dist[v][u] = len;
        return ;
    }
    void join(int u, int v)
    {
        add_edge(u, v, 0);
        return ;
    }
    lli eval(void)
    {
        lli min_span = 0;
        rep(i, 1, n) {
            vis[i] = false;
            min_cost[i] = 1;
        }
        vis[1] = true;
        rep(i, 1, n) {
            int p = 0;
            rep(j, 1, n)
                if (!vis[j] && dist[min_cost[j]][j] < dist[min_cost[
p]][p])
                    p = j;
            if (p == 0)
                break;
            min_span += dist[min_cost[p]][p];
            // printf("add_edge %d -> %d : %lld\n", p, min_cost[p],
            //        dist[p][min_cost[p]]);
            vis[p] = true;
            rep(j, 1, n)
                if (dist[p][j] < dist[min_cost[j]][j])
                    min_cost[j] = p;
        }
        return min_span;
    }
    void init(int n)
    {
        this->n = n;
        rep(i, 0, n)
            rep(j, 0, n)
                dist[i][j] = infinit;
        return ;
    }
} graph;
```

Tarjan 强连通分量

- 题目：poj1236, poj2186
- 依赖：

模板描述

给定 n 个点 m 条边的有向图 $G = (V, E)$，求 G 的强连通分量。

强连通分量：该子图内任意两点间总存在一条仅由子图内边构成的路径。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$

算法简述

记数组 dfn_i 为点 i 被 dfs 到的次序编号（时间戳），low_i 为 i 和 i 的子树能够追溯到的最早的堆栈中的节点的时间戳。维护一个堆栈用于储存要处理的强连通分量。具体做法见代码。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：无向边的数量上限。
- `int[] belong`：点 i 属于第 $belong_i$ 个强连通分量。
- `int[] bsize`：第 i 个强连通分量的大小为 $bsize_i$。
- `void add_edge(int u, int v)`：添加一条 $u \to v$ 的有向边。
- `int eval()`：求图 G 的强连通分量，并返回强连通分量的个数。
- `void init(int n)`：初始化点数为 n 的图。

```
class Tarjan
{
public:
    struct edge
    {
        int u, v;
        edge *next;
    } epool[maxm], *edges[maxn];
    int n, ecnt, dcnt, bcnt;
    stack<int> stk;
    int instk[maxn], dfn[maxn], low[maxn];
    int belong[maxn], bsize[maxn];
    void add_edge(int u, int v)
    {
        edge *p = &epool[++ecnt];
        p->u = u; p->v = v;
        p->next = edges[u]; edges[u] = p;
        return ;
    }
    void dfs(int p)
    {
```

```
        low[p] = dfn[p] = ++dcnt;
        stk.push(p);
        instk[p] = true;
        for (edge *ep = edges[p]; ep; ep = ep->next) {
            int q = ep->v;
            if (!dfn[q]) {
                dfs(q);
                if (low[q] < low[p])
                    low[p] = low[q];
            } else if (instk[q] && dfn[q] < low[p]) {
                low[p] = dfn[q];
            }
        }
        if (dfn[p] == low[p]) {
            bsize[++bcnt] = 0;
            int q = 0;
            do {
                q = stk.top();
                stk.pop();
                instk[q] = false;
                belong[q] = bcnt;
                bsize[bcnt] += 1;
            } while (q != p);
        }
        return ;
    }
    int eval(void)
    {
        while (!stk.empty())
            stk.pop();
        dcnt = bcnt = 0;  // dfs counter, component counter
        rep(i, 1, n) {
            dfn[i] = low[i] = 0;
            instk[i] = false;
            belong[i] = bsize[i] = 0;
        }
        rep(i, 1, n)
            if (!dfn[i])
                dfs(i);
        return bcnt;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        rep(i, 1, n)
            edges[i] = nullptr;
        return ;
    }
} graph;
```

SPFA 最短路

- 题目：poj3259
- 依赖：

模板描述

给定 n 个点 m 条边的有向图 $G = (V, E)$，$s \in V$ 以及每条边的长度，判定图是否存在负权回路，若无则求点 s 到 V 中任意一点的最短距离及满足距离最短的任意一条最短路径。

复杂度

- Space:
  - Worst Case: $O(n+m)$
  - Amortized: $O(n+m)$
  - Best Case: $O(n+m)$
- Time:
  - Worst Case: $O(nm)$
  - Amortized: $O(km)$
  - Best Case: $O(km)$

算法简述

松弛操作：对于边 $e(i,j) \in E$，$dist_j := min(dist_j, dist_i + len_e)$

写法类似 Dijkstra 算法，不使用堆优化，并保存一个 $qcnt$ 数组代表每个点进入队列的次数。另记一个数组 $inque$ 代表一个数是否在队列里。如果一个点重复入队超过 $n-1$ 次，则原图必存在负权回路。

通过比较当前松弛点的距离和队首的距离选择是推入队首还是队末的方法被称为前向星优化。注意在某些情况下该优化方法会被出题人针对并被卡掉，这时需去除前向星优化使用朴素方法。

获得最短路径的方法已在 Dijkstra 模板中记录，此处不予冗述。

一般地，前向星优化能够加速 1 到 10 倍不等，取决于数据强度。在无负权回路的图下，考虑到玄学常数 k，强烈建议使用堆优化 Dijkstra 而不是 SPFA。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：有向边的数量上限。
- `lli infinit`：规定的无限远距离。
- `lli[] dist`：对于每个点 $i \in [1, n]$，点 i 到源点 s 的距离。
- `edge[] from`：从源点到点 i 的最短路径上指向 i 的边的地址。
- `void add_edge(int u, int v, lli len)`：添加一条 $u \to v$，长度为 len 的有向边。

- `bool eval(int s)`：以 s 为源点，计算到 $[1,n]$ 所有点的最短路。同时若函数返回 $false$ 则该图包含负权回路，反之则不存在负权回路。
- `void init(int n)`：初始化点数为 n 的图。

```
class SPFA
{
public:
    struct edge
    {
        int u, v;
        lli len;
        edge *next;
    } epool[maxm], *edges[maxn], *from[maxn];
    int n, ecnt;
    lli dist[maxn];
    int qcnt[maxn], inque[maxn];
    void add_edge(int u, int v, lli len)
    {
        edge *p = &epool[++ecnt];
        p->u = u; p->v = v; p->len = len;
        p->next = edges[u]; edges[u] = p;
        return ;
    }
    bool eval(int s)
    {
        #define USE_SLF
        #ifdef USE_SLF
        deque<int> que;
        #else
        queue<int> que;
        #endif
        rep(i, 0, n) {
            qcnt[i] = 0;
            inque[i] = false;
            dist[i] = infinit;
            from[i] = nullptr;
        }
        dist[s] = 0;
        qcnt[s] += 1;
        inque[s] = true;
        #ifdef USE_SLF
        que.push_back(s);
        #else
        que.push(s);
        #endif
        while (!que.empty()) {
            int p = que.front();
            #ifdef USE_SLF
            que.pop_front();
            #else
            que.pop();
            #endif
            inque[p] = false;
            for (edge *ep = edges[p]; ep; ep = ep->next)
```

```
                if (dist[p] + ep->len < dist[ep->v]) {
                    dist[ep->v] = dist[p] + ep->len;
                    from[ep->v] = ep;
                    if (!inque[ep->v]) {
                        inque[ep->v] = true;
                        qcnt[ep->v] += 1;
                        if (qcnt[ep->v] >= n)
                            return false;
                        #ifdef USE_SLF
                        if (que.empty() || dist[ep->v] > dist[que.fr
ont()])
                            que.push_back(ep->v);
                        else
                            que.push_front(ep->v);
                        #else
                        que.push(ep->v);
                        #endif
                    }
                }
        }
        return true;
    }
    void init(int n)
    {
        this->n = n;
        ecnt = 0;
        rep(i, 1, n)
            edges[i] = nullptr;
        return ;
    }
} graph;
```

SPFA 费用流

- 题目：luogu3381
- 依赖：

模板描述

给定 n 个点 m 条边的有向图 $G = (V, E)$，每条边的流量和代价，给定源点 s 和汇点 t，求从点 s 到点 t 的最大流量，以及达成最大流量前提下的最小总花费。

复杂度

- Space:
  - Worst Case: $O(n + m)$
  - Amortized: $O(n + m)$
  - Best Case: $O(n + m)$
- Time:
  - Worst Case: $O(n^2 m)$
  - Amortized: $O(n^2 m)$

- Best Case: $O(n+m)$

算法简述

SPFA 费用流每次求一条代价最小的增广路，然后将这条增广路上的最大流量求出，并去掉这条流量。进一步地，可以使用多路增广，用类似 Dinic 的写法加速每次 BFS 增广的流量。我们甚至可以借用 Dijkstra 写法中的优先队列来优化 BFS 速度。注意这里 `set` 去重的效率并不如 `priority_queue` 直接推入。

调用方法

- `int maxn`：点的数量上限。
- `int maxm`：有向边的数量上限。
- `lli infinit`：规定的无限大流量。
- `void add_edge(int u, int v, lli flow, lli cost)`：添加一条 $u \to v$，流量为 $flow$，代价为 $cost$ 的有向边。
- `<lli, lli> eval()`：计算该有向图的最大流和对应的最小费用。
- `void init(int n, int s, int t)`：初始化点数为 n，源点为 s，汇点为 t 的图。

```
class SPFACostFlow
{
public:
    typedef pair<lli, int> pli;
    struct edge
    {
        int u, v;
        lli flow, cost;
        edge *next, *rev;
    } epool[maxm], *edges[maxn], *from[maxn];
    int n, s, t, ecnt;
    int vis[maxn];
    lli dist[maxn], height[maxn];
    void add_edge(int u, int v, lli flow, lli cost)
    {
        edge *p = &epool[++ecnt],
             *q = &epool[++ecnt];
        p->u = u; p->v = v; p->flow = flow; p->cost = cost;
        p->next = edges[u]; edges[u] = p;
        q->u = v; q->v = u; q->flow = 0; q->cost = - cost;
        q->next = edges[v]; edges[v] = q;
        p->rev = q; q->rev = p;
        return ;
    }
    bool spfa(void)
    {
        rep(i, 1, n)
            dist[i] = infinit;
        priority_queue<pli, vector<pli>, greater<pli>> pq;
        dist[s] = 0;
        pq.push(make_pair(dist[s], s));
```

```
        while (!pq.empty()) {
            pli pr = pq.top();
            int p = pr.second;
            pq.pop();
            if (dist[p] < pr.first)
                continue;
            for (edge *ep = edges[p]; ep; ep = ep->next)
                if (ep->flow > 0 && dist[p] + ep->cost +
                        height[p] - height[ep->v] < dist[ep->v]) {
                    dist[ep->v] = dist[p] + ep->cost + height[p] -
                                  height[ep->v];
                    from[ep->v] = ep;
                    pq.push(make_pair(dist[ep->v], ep->v));
                }
        }
        return dist[t] < infinit;
    }
    lli dfs(int p, lli flow, lli& rcost)
    {
        if (p == t || flow == 0)
            return flow;
        vis[p] = true;
        lli used = 0;
        for (edge *ep = edges[p]; ep; ep = ep->next)
            if (!vis[ep->v] && ep->flow > 0 && height[ep->v] ==
                    height[p] + ep->cost) {
                lli tmp = dfs(ep->v, min(flow - used, ep->flow), rco
st);

                used += tmp;
                ep->flow -= tmp;
                ep->rev->flow += tmp;
                rcost += tmp * ep->cost;
                if (used == flow)
                    break;
            }
        vis[p] = false;
        return used;
    }
    pair<lli, lli> eval(void)
    {
        memclr(height, n);
        lli rflow = 0, rcost = 0;
        while (spfa()) {
            rep(i, 1, n)
                height[i] = min(infinit, height[i] + dist[i]);
            lli tmp = dfs(s, infinit, rcost);
            if (tmp == 0)
                break;
            rflow += tmp;
        }
        return make_pair(rflow, rcost);
    }
    void init(int n, int s, int t)
    {
```

```
        this->n = n;
        this->s = s;
        this->t = t;
        ecnt = 0;
        memclr(edges, n);
        return ;
    }
} graph;
```

Splay 树

- 题目：
- 依赖：

模板描述

Splay

复杂度

- Space:
  - Worst Case: $O(n)$
  - Amortized: $O(n)$
  - Best Case: $O(n)$
- Time:
  - Worst Case: $O(n)$
  - Amortized: $O(\log n)$
  - Best Case: $O(\log n)$

算法简述

Splay（待补

调用方法

- `int maxn`：点的数量上限。
- `void init(int n)`：初始化点数为 n 的限制组。

```
class SplayTree
{
public:
    int n, root, ncnt;
    int ch[maxn][2], parent[maxn], size[maxn];
    lli val[maxn], vsm[maxn], vmn[maxn], vmx[maxn];
    lli lazyadd[maxn];
    bool lazyswp[maxn];
    #define lc(x) ch[x][0]
    #define rc(x) ch[x][1]
    #define par(x) parent[x]
    void push_down(int p)
    {
        rep(_, 0, 1)
```

```
        if (ch[p][_]) {
            lazyadd[ch[p][_]] += lazyadd[p];
            val[ch[p][_]] += lazyadd[p];
            vsm[ch[p][_]] += size[ch[p][_]] * lazyadd[p];
            vmn[ch[p][_]] += lazyadd[p];
            vmx[ch[p][_]] += lazyadd[p];
            lazyswp[ch[p][_]] ^= lazyswp[p];
        }
    if (lazyswp[p])
        swap(lc(p), rc(p));
    lazyadd[p] = 0;
    lazyswp[p] = false;
    return ;
}
void pull_up(int p)
{
    size[p] = size[lc(p)] + 1 + size[rc(p)];
    vsm[p] = vmn[p] = vmx[p] = val[p];
    rep(_, 0, 1)
        if (ch[p][_]) {
            vsm[p] += vsm[ch[p][_]];
            vmn[p] = min(vmn[p], vmn[ch[p][_]]);
            vmx[p] = max(vmx[p], vmx[ch[p][_]]);
        }
    return ;
}
int makenode(int q, lli v)
{
    int p = ++ncnt; n += 1;
    lc(p) = rc(p) = 0;
    par(p) = q;
    size[p] = 1;
    val[p] = vsm[p] = vmn[p] = vmx[p] = v;
    lazyadd[p] = 0;
    lazyswp[p] = false;
    return p;
}
void rotate(int p)
{
    int q = par(p), g = par(q);
    push_down(q);
    push_down(p);
    int x = p == rc(q);
    // relink connections
    ch[q][x] = ch[p][!x];
    if (ch[q][x]) par(ch[q][x]) = q;
    ch[p][!x] = q; par(q) = p;
    par(p) = g;
    if (g) ch[g][q == rc(g)] = p;
    pull_up(q);
    pull_up(p);
    return ;
}
int pre(int p)
```

```
{
    if (!lc(p)) {
        while (p == lc(par(p)))
            p = par(p);
        p = par(p);
    } else {
        p = lc(p);
        while (rc(p))
            p = rc(p);
    }
    return p;
}
int suc(int p)
{
    if (!rc(p)) {
        while (p == rc(par(p)))
            p = par(p);
        p = par(p);
    } else {
        p = rc(p);
        while (lc(p))
            p = lc(p);
    }
    return p;
}
void splay(int p, int t)
{
    for (int q = 0; (q = par(p)) && q != t; rotate(p))
        if (par(q) && par(q) != t)
            rotate((p == lc(q)) == (q == lc(par(q))) ? q : p);
    if (t == 0) root = p;
    return ;
}
int find(int x)
{
    int p = root;
    while (x > 0 && p) {
        push_down(p);
        if (x <= size[lc(p)]) {
            p = lc(p);
            continue;
        } x -= size[lc(p)];
        if (x <= 1) {
            return p;
        } x -= 1;
        p = rc(p);
    }
    return 0;
}
int find_bin_geq(lli v)
{
    // first p s.t. val[p] >= v
    int p = root;
    int q = 2;
```

```
    while (p) {
        push_down(p);
        if (val[p] == v) {
            return p;
        } else if (val[p] < v) {
            p = rc(p);
        } else if (val[p] > v) {
            q = val[p] < val[q] ? p : q;
            p = lc(p);
        }
    }
    return q;
}
int find_bin_leq(lli v)
{
    // last p. s.t. val[p] <= v
    int p = root;
    int q = 1;
    while (p) {
        push_down(p);
        if (val[p] == v) {
            return p;
        } else if (val[p] > v) {
            p = lc(p);
        } else if (val[p] < v) {
            q = val[p] > val[q] ? p : q;
            p = rc(p);
        }
    }
    return q;
}
void insert(int x, lli v)
{
    int lp = find(x + 1), rp = suc(lp);
    splay(rp, 0);
    splay(lp, root);
    int c = makenode(lp, v);
    rc(lp) = c;
    pull_up(lp);
    pull_up(rp);
    return ;
}
void insert_bin(lli v)
{
    int p = root;
    int q = 0;
    while (p) {
        push_down(p);
        if (v <= val[p]) {
            if (!lc(p)) {
                lc(p) = makenode(p, v);
                q = lc(p);
                break;
            }
```

```
                p = lc(p);
            } else {
                if (!rc(p)) {
                    rc(p) = makenode(p, v);
                    q = rc(p);
                    break;
                }
                p = rc(p);
            }
        }
        splay(q, 0);
        return ;
    }
    void remove(int l, int r)
    {
        int lp = find(l - 1 + 1), rp = find(r + 1 + 1);
        splay(rp, 0);
        splay(lp, root);
        int c = rc(lp);
        par(c) = 0;
        rc(lp) = 0;
        pull_up(lp);
        pull_up(rp);
        n -= r - l + 1;
        return ;
    }
    lli query(int l, int r, int mode)
    {
        // mode = 1: count, 2: sum, 3: min, 4: max
        if (l > r) {
            if (mode == 1)
                return 0;
            if (mode == 2)
                return 0;
            if (mode == 3)
                return infinit;
            if (mode == 4)
                return - infinit;
        }
        int lp = find(l - 1 + 1), rp = find(r + 1 + 1);
        splay(rp, 0);
        splay(lp, root);
        int p = rc(lp);
        if (mode == 1)
            return size[p];
        else if (mode == 2)
            return vsm[p];
        else if (mode == 3)
            return vmn[p];
        else if (mode == 4)
            return vmx[p];
        return 0;
    }
    lli query_bin(lli lb, lli rb, int mode)
```

```
{
    // mode = 1: count, 2: sum, 3: min, 4: max
    if (lb > rb) {
        if (mode == 1)
            return 0;
        if (mode == 2)
            return 0;
        if (mode == 3)
            return infinit;
        if (mode == 4)
            return - infinit;
    }
    int lp = find_bin_leq(lb - 1), rp = find_bin_geq(rb + 1);
    printf("found %d %d\n", lp, rp);
    splay(rp, 0);
    splay(lp, root);
    int p = rc(lp);
    if (mode == 1)
        return size[p];
    else if (mode == 2)
        return vsm[p];
    else if (mode == 3)
        return vmn[p];
    else if (mode == 4)
        return vmx[p];
    return 0;
}
void modify_add(int l, int r, lli v)
{
    int lp = find(l - 1 + 1), rp = find(r + 1 + 1);
    splay(rp, 0);
    splay(lp, root);
    int p = rc(lp);
    lazyadd[p] += v;
    val[p] += v;
    vsm[p] += size[p] * v;
    vmn[p] += v;
    vmx[p] += v;
    pull_up(lp);
    pull_up(rp);
    return ;
}
void modify_swp(int l, int r)
{
    int lp = find(l - 1 + 1), rp = find(r + 1 + 1);
    splay(rp, 0);
    splay(lp, root);
    int p = rc(lp);
    lazyswp[p] ^= 1;
    return ;
}
void init(void)
{
    n = ncnt = 0;
```

```
        root = makenode(0, - infinit);
        rc(root) = makenode(root, infinit);
        pull_up(rc(root));
        pull_up(root);
        return ;
    }
} splay;
```

Trie 字典树

- 题目：hdu1251
- 依赖：

模板描述

Trie

复杂度

- Space:
  - Worst Case: $O(n)$
  - Amortized: $O(n)$
  - Best Case: $O(n)$
- Time:
  - Worst Case: $O(n)$
  - Amortized: $O(\log n)$
  - Best Case: $O(\log n)$ 待补

算法简述

Trie 待补

调用方法

- `int maxn`：点的数量上限。
- `void init(int n)`：初始化点数为 n 的限制组。

```
class Trie
{
public:
    int ncnt, root;
    int ch[maxn][26];
    bool flag[maxn];
    int size[maxn], data[maxn];
    int make_node(void)
    {
        int p = ++ncnt;
        memclr(ch[p], 25);
        flag[p] = false;
        size[p] = data[p] = 0;
        return p;
```

```
    }
    void insert(char str[], int vdata = 0)
    {
        int p = root;
        for (int i = 0; str[i] != '\0'; i++) {
            int v = str[i] - 'a';
            size[p] += 1;
            if (!ch[p][v])
                ch[p][v] = make_node();
            p = ch[p][v];
        }
        size[p] += 1;
        flag[p] = true;
        data[p] = vdata;
        return ;
    }
    void remove(char str[])
    {
        int p = root;
        for (int i = 0; str[i] != '\0'; i++) {
            int v = str[i] - 'a';
            size[p] -= 1;
            p = ch[p][v];
        }
        size[p] -= 1;
        flag[p] = false;
        return ;
    }
    bool find(char str[])
    {
        int p = root;
        for (int i = 0; str[i] != '\0'; i++) {
            int v = str[i] - 'a';
            if (!ch[p][v])
                return false;
            p = ch[p][v];
        }
        // return anything you want here
        return false;
    }
    void init(void)
    {
        ncnt = 0;
        root = make_node();
        return ;
    }
} trie;
```

ACM-ICPC 模板整理

HDU_JustWe

2016 年 4 月 21 日

目录

1 动态规划

1.1 基于位运算的最长公共子序列

输入两个串 S 和 T，长度分别为 n_1 和 n_2，压 B 位，$ap[i][j]$ 表示字符 i 在 S 串的第 j 位是否存在，$\sum_{k=0}^{j} row[i][k]$ 表示 T 串前 i 位与 S 串前 j 位的 LCS。时间复杂度 $O(\frac{n_1 n_2}{B})$。

```
#include<cstdio>
typedef long long ll;
const int BIT=808,E=62;
int n1,n2,m,h,i,j,p,ans;char s[50000],t[50000];
struct Num{
  ll x[BIT];
  Num(){for(int i=0;i<BIT;i++)x[i]=0;}
  void set(int p){x[p/E]|=1LL<<(p%E);}
  Num operator&(Num b){
    Num c;
    for(int i=0;i<=m;i++)c.x[i]=x[i]&b.x[i];
    return c;
  }
  Num operator|(Num b){
    Num c;
    for(int i=0;i<=m;i++)c.x[i]=x[i]|b.x[i];
    return c;
  }
  Num operator^(Num b){
    Num c;
    for(int i=0;i<=m;i++)c.x[i]=x[i]^b.x[i];
    return c;
  }
  Num operator-(Num b){
    Num c;
    for(int i=0;i<=m;i++)c.x[i]=x[i]-b.x[i];
    for(int i=0;i<m;i++)if(c.x[i]<0)c.x[i]+=(1LL<<E),c.x[i+1]--;
    return c;
  }
  void shl(){
    ll o=1,p;
    for(int i=0;i<=m;o=p,i++){
      p=x[i]&(1LL<<h),(x[i]<<=1)&=~(1LL<<(h+1));
      if(o)x[i]|=1;
    }
  }
}ap[4],x,row[2];
int hash(int x){
  if(x=='A')return 0;
  if(x=='C')return 1;
  if(x=='G')return 2;
  return 3;
}
int main(){
  scanf("%d%d%s%s",&n1,&n2,s,t);
  for(m=(n1-1)/E,h=(m?E:n1)-1;i<n1;i++)ap[hash(s[i])].set(i);
  for(i=0;i<n2;i++){
    p^=1;
    x=ap[hash(t[i])]|row[p^1];
```

```
    row[p^1].shl();
    row[p]=x&((x-row[p^1])^x);
  }
  for(i=m;~i;i--)for(j=h;~j;j--)if(row[p].x[i]&(1LL<<j))ans++;
  printf("%d",ans);
}
```

1.2 决策单调且不要求在线时的糙快猛优化方法

$[l,r]$ 表示要 DP 的区间，$[dl,dr]$ 表示可用的最优决策取值区间，时间复杂度 $O(n\log n)$。

```
void dp(int l,int r,int dl,int dr){
  if(l>r)return;
  int m=(l+r)>>1,i,dm;
  for(i=dl;i<=dr;i++)if(i更新f[m]更优)dm=i;
  用dm更新f[m];
  dp(l,m-1,dl,dm),dp(m+1,r,dm,dr);
}
```

1.3 悬线法

输入 $n\times m$ 的 01 矩阵，求面积最大的全为 1 的子矩阵，时间复杂度 $O(nm)$。

```
#include<cstdio>
#define N 1001
int n,m,i,j,ans,l[N],r[N],h[N],lmax,rmax,a[N][N];
int main(){
  for(scanf("%d%d",&n,&m),i=1;i<=n;i++)for(j=1;j<=m;j++)scanf("%d",&a[i][j]);
  for(i=1;i<=m;i++)l[i]=1,r[i]=m;
  for(i=1;i<=n;i++){
    for(lmax=j=1;j<=m;j++)if(a[i][j]){
      h[j]++;
      if(lmax>l[j])l[j]=lmax;
    }else h[j]=0,l[j]=1,r[j]=m,lmax=j+1;
    for(rmax=j=m;j;j--)if(a[i][j]){
      if(rmax<r[j])r[j]=rmax;
      if((r[j]-l[j]+1)*h[j]>ans)ans=(r[j]-l[j]+1)*h[j];
    }else rmax=j-1;
  }
  printf("%d",ans);
}
```

1.4 插头 DP

以三进制表示轮廓线上插头的状态，0 表示无插头，1 表示左括号，2 表示右括号。时间复杂度 $O(nm3^m)$。

代码中点表示这个块可以通过，横表示这个块只可以左右通过，竖表示这个块只可以上下通过，井表示这个块不能通过，最后求出的是把所有可以通过的块都经过且只经过一次并回到原地的方案数。

```
#include<cstdio>
#define now f[j]
#define pre f[j-1]
typedef long long ll;
int n,m,x,y,i,j,k,h,S,e,pow[14],q[41836],p[14][41840],st[14],can;
int lasti,lastj,firsti,firstj,hash[1594324];
ll ans,f[14][41836];
char ch[14][14];
int bit(int x,int i){return x/pow[i]%3;}
void up(ll&a,ll b){a+=b;}
int main(){
  scanf("%d%d",&n,&m);
  for(i=1;i<=n;i++)for(scanf("%s",ch[i]+1),j=1;j<=m;j++)if(ch[i][j]!='#'){
    lasti=i,lastj=j;
    if(!firsti)firsti=i,firstj=j;
  }
  for(pow[0]=i=1;i<=m+1;i++)pow[i]=pow[i-1]*3;
  S=pow[m+1];
  for(i=0;i<S;i++){
    can=1;st[0]=0;
    for(j=0;j<=m;j++){
      k=bit(i,j);
      if(k==2){
        if(!st[0]){can=0;break;}
        p[st[st[0]]][q[0]+1]=j;p[j][q[0]+1]=st[st[0]];
        --st[0];
      }
      if(k==1)st[++st[0]]=j;
    }
    if(can&&!st[0])q[hash[i]=++q[0]]=i;
  }
  for(i=1;i<=n;i++){
    for(k=0;k<=q[0];k++)f[0][k]=0;
    for(k=1;k<=q[0];k++)
      if(f[m][k]&&!bit(q[k],m))
        f[0][hash[(q[k]-bit(q[k],m)*pow[m])*3]]=f[m][k];
    for(j=1;j<=m;j++)for(k=0;k<=q[0];k++)f[j][k]=0;
    for(j=1;j<=m;j++){
      if(ch[i][j]=='.'&&i==firsti&&j==firstj)up(now[hash[pow[j-1]+pow[j]*2]],1);
      for(h=1;h<=q[0];h++){
        k=q[h];
        if(!pre[h])continue;
        x=bit(k,j-1),y=bit(k,j),e=k-x*pow[j-1]-y*pow[j];
        if(!x&&!y){
          if(ch[i][j]!='-'&&ch[i][j]!='|'){
            if(ch[i][j]=='.')up(now[hash[e+pow[j-1]+2*pow[j]]],pre[h]);
            if(ch[i][j]=='#')up(now[h],pre[h]);
          }
        }else if(!x){
          if(ch[i][j]!='#'&&ch[i][j]!='-'){
            up(now[hash[e+y*pow[j-1]]],pre[h]);
            if(ch[i][j]=='.')up(now[hash[e+y*pow[j]]],pre[h]);
          }
        }else if(!y){
          if(ch[i][j]!='#'&&ch[i][j]!='|'){
            if(ch[i][j]=='.')up(now[hash[e+x*pow[j-1]]],pre[h]);
```

```
          up(now[hash[e+x*pow[j]]],pre[h]);
        }
      }else if(ch[i][j]=='.'){
        if(x==1&&y==2&&!e&&i==lasti&&j==lastj)up(ans,pre[h]);
        else if(x==2&&y==1)up(now[hash[e]],pre[h]);
        else if(x==y){
          int t=e-bit(e,p[j-1][h])*pow[p[j-1][h]]
                -bit(e,p[j][h])*pow[p[j][h]]
                +pow[p[j][h]]+2*pow[p[j-1][h]];
          up(now[hash[t]],pre[h]);
        }
      }
    }
  }
}
printf("%lld",ans);
}
```

1.5 整数划分

$f[i][j]$ 表示选了 i 种不同的数字，总和为 j 的方案数。

$f[i][j]=f[i-1][j-1]+f[i][j-i]$，此式子的意义为要么新选一个 1，要么之前选的都增加 1。若每种数字最多选一个，那么有 $f[i][j]=f[i-1][j-i]+f[i][j-i]$。

对于求把 n 划分成若干整数的和的方案数，设 $g[i]$ 表示 $n=i$ 时的答案，代码如下，时间复杂度 $O(n\sqrt{n})$。

```
#include<cstdio>
const int N=731,P=1000000007;
int n,m,i,j,f[N+1],g[200001];
int main(){
  for(f[1]=1,f[2]=2,f[3]=5,f[4]=7,i=5;i<N;i++)f[i]=3+2*f[i-2]-f[i-4];
  for(scanf("%d",&n),g[0]=i=1;i<=n;i++)
    for(j=1;f[j]<=i;j++)
      if((j+1)>>1&1)g[i]=(g[i]+g[i-f[j]])%P;
      else g[i]=(g[i]-g[i-f[j]])%P;
}
```

2 莫队算法

2.1 普通莫队算法

```
#include<cstdio>
#include<algorithm>
#define N 50010
using namespace std;
int n,m,lim,i,l,r,ans[N];
struct Q{int l,r,p;}q[N];
bool cmp(const Q&a,const Q&b){return pos[a.l]==pos[b.l]?a.r<b.r:pos[a.l]<pos[b.l];}
int main(){
  scanf(”%d%d”,&n,&m);
  while(lim*lim<n)lim++;
  for(i=1;i<=n;i++)pos[i]=(i-1)/lim+1;
  for(i=1;i<=m;i++)read(q[i].l),read(q[i].r),q[i].p=i;
  sort(q+1,q+m+1,cmp);
  for(i=l=1,r=0;i<=m;i++){
    int L=q[i].l,R=q[i].r;
    if(r<R){for(r++;r<=R;r++)add(r);r--;}
    if(r>R)for(;r>R;r--)del(r);
    if(l<L)for(;l<L;l++)del(l);
    if(l>L){for(l--;l>=L;l--)add(l);l++;}
    ans[q[i].p]=now;
  }
  for(i=1;i<=m;i++)printf(”%d\n”,ans[i]);
}
```

2.2 树上莫队算法

按 DFS 序分块，查询的时候需要额外考虑 lca。

```
#include<cstdio>
#include<algorithm>
#include<cmath>
#define N 100010
#define K 17
using namespace std;
struct P{int l,r,z,id;}Q[N];
int lim,pos[N<<1],l,r,c[N],g[N],v[N<<1],nxt[N<<1],ed;
int n,m,i,j,x,y,z,loc[N<<1],dfn,st[N],en[N],d[N],f[N][18];
int ans[N],cnt[N],sum;bool vis[N];
bool cmp(const P&a,const P&b){return pos[a.l]==pos[b.l]?a.r<b.r:pos[a.l]<pos[b.l];}
void add(int x,int y){v[++ed]=y;nxt[ed]=g[x];g[x]=ed;}
void dfs(int x){
  for(int i=vis[loc[st[x]=++dfn]=x]=1;i<=K;i++)f[x][i]=f[f[x][i-1]][i-1];
  for(int i=g[x];i;i=nxt[i])if(!vis[v[i]])d[v[i]]=d[f[v[i]][0]=x]+1,dfs(v[i]);
  loc[en[x]=++dfn]=x;
}
int lca(int x,int y){
  if(x==y)return x;
  if(d[x]<d[y])swap(x,y);
  for(int i=K;~i;i--)if(d[f[x][i]]>=d[y])x=f[x][i];
  if(x==y)return x;
```

```
  for(int i=K;~i;i--)if(f[x][i]!=f[y][i])x=f[x][i],y=f[y][i];
  return f[x][0];
}
void deal(int x){
  if(!vis[x]){if(!(--cnt[c[x]]))sum--;}else if(!(cnt[c[x]]++))sum++;
  vis[x]^=1;
}
int main(){
  for(read(n),read(m),i=1;i<=n;i++)read(c[i]);
  for(i=1;i<=n;i++)read(x),read(y),add(x,y),add(y,x);
  dfs(d[1]=1),lim=(int)sqrt(n*2+0.5);
  for(i=1;i<=dfn;i++)pos[i]=(i-1)/lim+1;
  for(i=1;i<=m;i++){
    read(x),read(y);Q[i].id=i;
    if(st[x]>st[y])swap(x,y);
    z=lca(x,y);
    if(z==x)Q[i].l=st[x],Q[i].r=st[y];
    else Q[i].l=en[x],Q[i].r=st[y],Q[i].z=z;
  }
  for(sort(Q+1,Q+m+1,cmp),i=1,l=1,r=0;i<=m;i++){
    if(r<Q[i].r){for(r++;r<=Q[i].r;r++)deal(loc[r]);r--;}
    if(r>Q[i].r)for(;r>Q[i].r;r--)deal(loc[r]);
    if(l<Q[i].l)for(;l<Q[i].l;l++)deal(loc[l]);
    if(l>Q[i].l){for(l--;l>=Q[i].l;l--)deal(loc[l]);l++;}
    if(Q[i].z)deal(Q[i].z);
    ans[Q[i].id]=now;
    if(Q[i].z)deal(Q[i].z);
  }
  for(i=1;i<=m;i++)printf("%d\n",ans[i]);
}
```

2.3 树上带修改莫队算法

将 DFS 序分成 $n^{\frac{1}{3}}$ 块，枚举两块，然后按时间处理操作，时间复杂度 $O(n^{\frac{5}{3}})$。

```
#include<cstdio>
#include<algorithm>
#define N 50010
#define K 15
using namespace std;
struct Que{int l,r,t,z;}ask[N];
int ans[N];
int n,m,q,i,j,k,x,y,z,f[N][16],d[N],B,nl,nr,l,r,vis[N],C[N],c[N];
int T,mx[N],my[N],op[N];
int st[N],en[N],dfn[N<<1],pos[N<<1],post;
int g[N],v[N<<1],nxt[N<<1],ed,que[50][50],fin[50][50];
int ap[N],h[N],full[N],now[N];
void add(int x,int y){v[++ed]=y;nxt[ed]=g[x];g[x]=ed;}
void dfs(int x,int pre){
  dfn[st[x]=++post]=x;
  int i=1;
  for(f[x][0]=pre;i<=K;i++)f[x][i]=f[f[x][i-1]][i-1];
  for(i=g[x];i;i=nxt[i])if(v[i]!=pre)d[v[i]]=d[x]+1,dfs(v[i],x);
  dfn[en[x]=++post]=x;
}
```

```
int lca(int x,int y){
  if(x==y)return x;
  if(d[x]<d[y])swap(x,y);
  for(int i=K;~i;i--)if(d[f[x][i]]>=d[y])x=f[x][i];
  if(x==y)return x;
  for(int i=K;~i;i--)if(f[x][i]!=f[y][i])x=f[x][i],y=f[y][i];
  return f[x][0];
}
inline void addq(int x,int y,int z){
  v[++ed]=z;nxt[ed]=0;
  if(!que[x][y])que[x][y]=ed;else nxt[fin[x][y]]=ed;
  fin[x][y]=ed;
}
void deal(int x){
  if(c[x]<=n){
    if(vis[x]){
      ap[c[x]]--;
      if(!ap[c[x]])now[pos[c[x]]]--;
    }else{
      if(!ap[c[x]])now[pos[c[x]]]++;
      ap[c[x]]++;
    }
  }
  vis[x]^=1;
}
int askmex(){
  for(int i=0;;i++)if(full[i]>now[i])
    for(int j=h[i];;j++)if(!ap[j])return j;
}
int main(){
  read(n);read(q);
  for(i=1;i<=n;i++)read(C[i]);
  for(i=1;i<n;i++)read(x),read(y),add(x,y),add(y,x);
  dfs(d[1]=1,ed=0);
  while(B*B*B<post)B++;B*=B;
  for(i=1;i<=post;i++)pos[i]=(i-1)/B+1;m=pos[post];
  for(i=1;i<=q;i++){
    read(op[i]);read(x);read(y);
    if(!op[i])mx[++T]=x,my[T]=y;
    else{
      if(st[x]>st[y])swap(x,y);
      z=lca(x,y);
      if(z==x)nl=st[x],nr=st[y];else nl=en[x],nr=st[y];
      ask[i].t=T;
      ask[i].l=nl;ask[i].r=nr;
      if(z!=x)ask[i].z=z;
      addq(pos[nl],pos[nr],i);
    }
  }
  for(B=1;B*B<=n;B++);
  for(i=1;i<=n;i++)pos[i]=(i-1)/B+1;
  for(i=0;i<=n;i++)full[pos[i]]++;
  for(i=n;i;i--)h[pos[i]]=i;
  for(i=1;i<=m;i++)for(j=i;j<=m;j++)if(que[i][j]){
    for(k=0;k<=n;k++)c[k]=C[k],vis[k]=ap[k]=0;
    for(k=0;k<=pos[n];k++)now[k]=0;
    T=1;l=(i-1)*B+1;r=l-1;
```

```
    for(k=que[i][j];k;k=nxt[k]){
      if(r<ask[v[k]].r){for(r++;r<=ask[v[k]].r;r++)deal(dfn[r]);r--;}
      if(r>ask[v[k]].r)for(;r>ask[v[k]].r;r--)deal(dfn[r]);
      if(l<ask[v[k]].l)for(;l<ask[v[k]].l;l++)deal(dfn[l]);
      if(l>ask[v[k]].l){for(l--;l>=ask[v[k]].l;l--)deal(dfn[l]);l++;}
      while(T<=ask[v[k]].t){
        bool flag=(ask[v[k]].l<=st[mx[T]]&&st[mx[T]]<=ask[v[k]].r)
                  ^(ask[v[k]].l<=en[mx[T]]&&en[mx[T]]<=ask[v[k]].r);
        if(flag)deal(mx[T]);
        c[mx[T]]=my[T];
        if(flag)deal(mx[T]);
        T++;
      }
      if(ask[v[k]].z)deal(ask[v[k]].z);
      ans[v[k]]=askmex();
      if(ask[v[k]].z)deal(ask[v[k]].z);
    }
  }
  for(i=1;i<=q;i++)if(op[i])printf("%d\n",ans[i]);
}
```

2.4 二维莫队算法

二维莫队算法，将矩阵横着分成 $\sqrt{n}$ 份，竖着分成 $\sqrt{m}$ 份，一共得到 $\sqrt{nm}$ 块，从左往右，从上到下编号。对于询问，以左上角所在块为第一关键字，右下角所在块为第二关键字排序，然后暴力转移。时间复杂度 $O(qn^{\frac{3}{2}})$。

```
#include<cstdio>
#include<algorithm>
using namespace std;
const int N=210,M=100010;
int n,m,Q,sn,sm,i,j,a[N][N],pos[N][N];
int X0,X1,Y0,Y1,l0,r0,l1,r1,now,ans[M];
struct P{int a,b,c,d,p;}q[M];
bool cmp(const P&a,const P&b){
  return pos[a.a][a.c]==pos[b.a][b.c]?
         pos[a.b][a.d]<pos[b.b][b.d]:pos[a.a][a.c]<pos[b.a][b.c];
}
int main(){
  scanf("%d%d",&n,&m);
  for(i=1;i<=n;i++)for(j=1;j<=m;j++)scanf("%d",&a[i][j]);
  while(sn*sn<n)sn++;
  while(sm*sm<m)sm++;
  for(i=1;i<=n;i++)for(j=1;j<=m;j++)pos[i][j]=i/sn*n+j/sm;
  for(scanf("%d",&Q),i=1;i<=Q;i++){
    scanf("%d%d%d%d",&X0,&Y0,&X1,&Y1);
    if(X0>X1)swap(X0,X1);
    if(Y0>Y1)swap(Y0,Y1);
    q[i].a=X0,q[i].b=X1,q[i].c=Y0,q[i].d=Y1,q[i].p=i;
  }
  for(sort(q+1,q+Q+1,cmp),i=l0=l1=1,r0=r1=0;i<=Q;i++){
    int L0=q[i].a,R0=q[i].b,L1=q[i].c,R1=q[i].d;
    if(r0<R0){for(r0++;r0<=R0;r0++)for(j=l1;j<=r1;j++)add(a[r0][j]);r0--;}
    if(r0>R0)for(;r0>R0;r0--)for(j=l1;j<=r1;j++)del(a[r0][j]);
```

```
        if(l0<L0)for(;l0<L0;l0++)for(j=l1;j<=r1;j++)del(a[l0][j]);
        if(l0>L0){for(l0--;l0>=L0;l0--)for(j=l1;j<=r1;j++)add(a[l0][j]);l0++;}
        if(r1<R1){for(r1++;r1<=R1;r1++)for(j=l0;j<=r0;j++)add(a[j][r1]);r1--;}
        if(r1>R1)for(;r1>R1;r1--)for(j=l0;j<=r0;j++)del(a[j][r1]);
        if(l1<L1)for(;l1<L1;l1++)for(j=l0;j<=r0;j++)del(a[j][l1]);
        if(l1>L1){for(l1--;l1>=L1;l1--)for(j=l0;j<=r0;j++)add(a[j][l1]);l1++;}
        ans[q[i].p]=now;
    }
    for(i=1;i<=Q;i++)printf("%d\n",ans[i]);
}
```

3 数据结构

3.1 Hash

3.1.1 Hash 表

```
const int M=262143;
struct E{int v;E*nxt;}*g[M+1],pool[N],*cur=pool,*p;int vis[M+1];
void ins(int v){
  int u=v&M;
  if(vis[u]<T)vis[u]=T,g[u]=NULL;
  for(p=g[u];p;p=p->nxt)if(p->v==v)return;
  p=cur++;p->v=v;p->nxt=g[u];g[u]=p;
}
int ask(int v){
  int u=v&M;
  if(vis[u]<T)return 0;
  for(p=g[u];p;p=p->nxt)if(p->v==v)return 1;
  return 0;
}
void init(){T++,cur=pool;}
```

3.1.2 字符串 Hash

```
const int N=20010,P=31,D=1000173169,M=262143;
int n,i,pow[N],f[N];char a[N];
int hash(int l,int r){return(ll)(f[r]-(ll)f[l-1]*pow[r-l+1]%D+D)%D;}
int main(){
  scanf("%d%s",&n,a+1);
  for(pow[0]=i=1;i<=n;i++)pow[i]=(ll)pow[i-1]*P%D;
  for(i=1;i<=n;i++)f[i]=(ll)((ll)f[i-1]*P+a[i])%D;
}
```

3.1.3 矩阵 Hash

找出某个 $x \times y$ 的矩阵在某个 $n \times m$ 的矩阵中的所有出现位置。首先对于每个位置，求出它开始长度为 y 的横行的 hash 值，然后对于 hash 值再求一次竖列的 hash 值即可。

```
#include<cstdio>
#include<algorithm>
#define N 1010
typedef unsigned long long ll;
const ll D1=197,D2=131;
int n,m,x,y,i,j,ans,t,cnt;
char a[N][N];
ll pow1[N],pow2[N],h[N][N],tmp;
struct P{
  int x,y;ll z;
  P(){}
  P(int _x,int _y,ll _z){x=_x,y=_y,z=_z;}
}q[N*N],fin[N];
bool cmp(const P&a,const P&b){return a.z<b.z;}
```

```
bool cmp2(const P&a,const P&b){return a.x==b.x?a.y<b.y:a.x<b.x;}
int main(){
  scanf("%d%d",&n,&m);gets(a[0]);
  for(i=1;i<=n;i++)gets(a[i]+1);
  scanf("%d%d",&x,&y);
  for(pow1[0]=pow2[0]=i=1;i<=n||i<=m;i++){pow1[i]=pow1[i-1]*D1,pow2[i]=pow2[i-1]*D2;
  for(i=1;i<=n;i++){
    for(tmp=0,j=1;j<y;j++)tmp=tmp*D1+a[i][j],h[i][j]=0;
    for(j=y;j<=m;j++)h[i][j]=tmp=tmp*D1-pow1[y]*a[i][j-y]+a[i][j];
  }
  for(t=0,i=y;i<=m;i++){
    for(tmp=0,j=1;j<x;j++)tmp=tmp*D2+h[j][i];
    for(j=x;j<=n;j++)q[++t]=P(j-x+1,i-y+1,tmp=tmp*D2-pow2[x]*h[j-x][i]+h[j][i]);
  }
  for(std::sort(q+1,q+t+1,cmp),j=1,i=2;i<=t;i++)if(q[i-1].z!=q[i].z){
    if(i-j>ans)ans=i-j,tmp=q[j].z;
    j=i;
  }
  if(t-j+1>ans)ans=t-j+1,tmp=q[t].z;
  printf("%d %d\n",x,y);
  for(i=1;i<=t;i++)if(q[i].z==tmp)fin[++cnt]=P(q[i].x,q[i].y,0);
  std::sort(fin+1,fin+cnt+1,cmp2);
  for(i=0;i<x;puts(""),i++)for(j=0;j<y;j++)putchar(a[fin[1].x+i][fin[1].y+j]);
  for(printf("%d\n",cnt),i=1;i<=cnt;i++)printf("%d %d\n",fin[i].x,fin[i].y);
}
```

3.2 树状数组区间修改区间查询

维护一个序列 $b[i]$，一开始都是 0，支持以下操作：

1. 把区间 $[x,y]$ 内的 $b[i]$ 加上 $a[i]$。

2. 查询区间 $[x,y]$ 内 $b[i]$ 的和。

代码中 s 为 a 的前缀和。

```
int n,m,i,op,x,y;
struct BIT{
  int n,s[N],a[N];ll b[N];
  void init(int x){n=x;for(int i=1;i<=n;i++)a[i]=b[i]=s[i]=0;}
  void modify(int x,int p){for(int i=x;i<=n;i+=i&-i)a[i]+=p,b[i]+=p*s[x-1];}
  ll ask(int x){
    int t0=0;ll t1=0;
    for(int i=x;i;i-=i&-i)t0+=a[i],t1+=b[i];
    return 1LL*s[x]*t0-t1;
  }
}T;
int main(){
  scanf("%d",&n);
  for(i=1;i<=n;i++)scanf("%d",&T.s[i]),T.s[i]+=T.s[i-1];
  scanf("%d",&m);
  while(m--){
    scanf("%d%d%d",&op,&x,&y);
    if(op==1)T.modify(x,1),T.modify(y+1,-1);
    else printf("%lld\n",T.ask(y)-T.ask(x-1));
  }
}
```

3.3 K-D Tree

```
#include<cstdio>
#include<algorithm>
#define N 200010
int n,i,id[N],root,cmp_d;
struct node{int d[2],l,r,Max[2],Min[2],val,sum,f;}t[N];
bool cmp(const node&a,const node&b){return a.d[cmp_d]<b.d[cmp_d];}
void umax(int&a,int b){if(a<b)a=b;}
void umin(int&a,int b){if(a>b)a=b;}
void up(int x){
  if(t[x].l){
    umax(t[x].Max[0],t[t[x].l].Max[0]);
    umin(t[x].Min[0],t[t[x].l].Min[0]);
    umax(t[x].Max[1],t[t[x].l].Max[1]);
    umin(t[x].Min[1],t[t[x].l].Min[1]);
  }
  if(t[x].r){
    umax(t[x].Max[0],t[t[x].r].Max[0]);
    umin(t[x].Min[0],t[t[x].r].Min[0]);
    umax(t[x].Max[1],t[t[x].r].Max[1]);
    umin(t[x].Min[1],t[t[x].r].Min[1]);
  }
}
int build(int l,int r,int D,int f){
  int mid=(l+r)>>1;
  cmp_d=D,std::nth_element(t+l+1,t+mid+1,t+r+1,cmp);
  id[t[mid].f]=mid;
  t[mid].f=f;
  t[mid].Max[0]=t[mid].Min[0]=t[mid].d[0];
  t[mid].Max[1]=t[mid].Min[1]=t[mid].d[1];
  t[mid].val=t[mid].sum=0;
  if(l!=mid)t[mid].l=build(l,mid-1,!D,mid);else t[mid].l=0;
  if(r!=mid)t[mid].r=build(mid+1,r,!D,mid);else t[mid].r=0;
  return up(mid),mid;
}
//输入的第x个点的权值增加p
void change(int x,int p){for(t[x=id[x]].val+=p;x;x=t[x].f)t[x].sum+=p;}
/*
估价函数:
欧几里得距离下界:
sqr(max(max(X-x.Max[0],x.Min[0]-X),0))+sqr(max(max(Y-x.Max[1],x.Min[1]-Y),0))
曼哈顿距离下界:
max(x.Min[0]-X,0)+max(X-x.Max[0],0)+max(x.Min[1]-Y,0)+max(Y-x.Max[1],0)
欧几里得距离上界:
max(sqr(X-x.Min[0]),sqr(X-x.Max[0]))+max(sqr(Y-x.Min[1]),sqr(Y-x.Max[1])
曼哈顿距离上界:
max(abs(X-x.Max[0]),abs(x.Min[0]-X))+max(abs(Y-x.Max[1]),abs(x.Min[1]-Y))
*/
//查询矩形范围内所有点的权值和
void ask(int x){
  if(t[x].Min[0]>X2||t[x].Max[0]<X1||t[x].Min[1]>Y2||t[x].Max[1]<Y1)return;
  if(t[x].Min[0]>=X1&&t[x].Max[0]<=X2&&t[x].Min[1]>=Y1&&t[x].Max[1]<=Y2){
    k+=t[x].sum;
    return;
  }
```

```
  if(t[x].d[0]>=X1&&t[x].d[0]<=X2&&t[x].d[1]>=Y1&&t[x].d[1]<=Y2)k+=t[x].val;
  if(t[x].l)ask(t[x].l);
  if(t[x].r)ask(t[x].r);
}
int main(){
  while(~scanf(”%d”,&n)){
    for(i=1;i<=n;i++){
      scanf(”%d%d”,&x,&y);
      t[i].d[0]=x,t[i].d[1]=y,t[i].f=i;
    }
    root=build(1,n,0,0);
  }
  return 0;
}
```

3.4 Link-Cut Tree

```
int f[N],son[N][2],val[N],sum[N],tmp[N];bool rev[N];
bool isroot(int x){return !f[x]||son[f[x]][0]!=x&&son[f[x]][1]!=x;}
void rev1(int x){if(!x)return;swap(son[x][0],son[x][1]);rev[x]^=1;}
void pb(int x){if(rev[x])rev1(son[x][0]),rev1(son[x][1]),rev[x]=0;}
void up(int x){
  sum[x]=val[x];
  if(son[x][0])sum[x]+=sum[son[x][0]];
  if(son[x][1])sum[x]+=sum[son[x][1]];
}
void rotate(int x){
  int y=f[x],w=son[y][1]==x;
  son[y][w]=son[x][w^1];
  if(son[x][w^1])f[son[x][w^1]]=y;
  if(f[y]){
    int z=f[y];
    if(son[z][0]==y)son[z][0]=x;else if(son[z][1]==y)son[z][1]=x;
  }
  f[x]=f[y];f[y]=x;son[x][w^1]=y;up(y);
}
void splay(int x){
  int s=1,i=x,y;tmp[1]=i;
  while(!isroot(i))tmp[++s]=i=f[i];
  while(s)pb(tmp[s--]);
  while(!isroot(x)){
    y=f[x];
    if(!isroot(y)){if((son[f[y]][0]==y)^(son[y][0]==x))rotate(x);else rotate(y);}
    rotate(x);
  }
  up(x);
}
void access(int x){for(int y=0;x;y=x,x=f[x])splay(x),son[x][1]=y,up(x);}
int root(int x){access(x);splay(x);while(son[x][0])x=son[x][0];return x;}
void makeroot(int x){access(x);splay(x);rev1(x);}
void link(int x,int y){makeroot(x);f[x]=y;access(x);}
void cutf(int x){access(x);splay(x);f[son[x][0]]=0;son[x][0]=0;up(x);}
void cut(int x,int y){makeroot(x);cutf(y);}
int ask(int x,int y){makeroot(x);access(y);splay(y);return sum[y];}
```

3.5 Top Tree

```
const int N=100010*2,inf=~0U>>1;
struct tag{
int a,b;//ax+b
tag(){a=1,b=0;}
tag(int x,int y){a=x,b=y;}
bool ex(){return a!=1||b;}
tag operator+(const tag&x){return tag(a*x.a,b*x.a+x.b);}
};
int atag(int x,tag y){return x*y.a+y.b;}
struct data{
int sum,minv,maxv,size;
data(){sum=size=0,minv=inf,maxv=-inf;}
data(int x){sum=minv=maxv=x,size=1;}
data(int a,int b,int c,int d){sum=a,minv=b,maxv=c,size=d;}
data operator+(const data&x){
  return data(sum+x.sum,min(minv,x.minv),max(maxv,x.maxv),size+x.size);
}
};
data operator+(const data&a,const tag&b){
  return a.size?data(a.sum*b.a+a.size*b.b,atag(a.minv,b),atag(a.maxv,b),a.size):a;
}
//son:0-1：重链儿子，2-3：AAA 树儿子
int f[N],son[N][4],a[N],tot,rt,rub,ru[N];bool rev[N],in[N];
int val[N];
data csum[N],tsum[N],asum[N];
tag ctag[N],ttag[N];
bool isroot(int x,int t){
  if(t)return !f[x]||!in[f[x]]||!in[x];
  return !f[x]||(son[f[x]][0]!=x&&son[f[x]][1]!=x)||in[f[x]]||in[x];
}
void rev1(int x){
  if(!x)return;
  swap(son[x][0],son[x][1]);rev[x]^=1;
}
void tagchain(int x,tag p){
  if(!x)return;
  csum[x]=csum[x]+p;
  asum[x]=csum[x]+tsum[x];
  val[x]=atag(val[x],p);
  ctag[x]=ctag[x]+p;
}
void tagtree(int x,tag p,bool t){
  if(!x)return;
  tsum[x]=tsum[x]+p;
  ttag[x]=ttag[x]+p;
  if(!in[x]&&t)tagchain(x,p);else asum[x]=csum[x]+tsum[x];
}
void pb(int x){
  if(!x)return;
  if(rev[x])rev1(son[x][0]),rev1(son[x][1]),rev[x]=0;
  if(!in[x]&&ctag[x].ex()){
    tagchain(son[x][0],ctag[x]);
    tagchain(son[x][1],ctag[x]);
    ctag[x]=tag();
```

```
  }
  if(ttag[x].ex()){
    tagtree(son[x][0],ttag[x],0),tagtree(son[x][1],ttag[x],0);
    tagtree(son[x][2],ttag[x],1),tagtree(son[x][3],ttag[x],1);
    ttag[x]=tag();
  }
}
void up(int x){
  tsum[x]=data();
  for(int i=0;i<2;i++)if(son[x][i])tsum[x]=tsum[x]+tsum[son[x][i]];
  for(int i=2;i<4;i++)if(son[x][i])tsum[x]=tsum[x]+asum[son[x][i]];
  if(in[x]){
    csum[x]=data();
    asum[x]=tsum[x];
  }else{
    csum[x]=data(val[x]);
    for(int i=0;i<2;i++)if(son[x][i])csum[x]=csum[x]+csum[son[x][i]];
    asum[x]=csum[x]+tsum[x];
  }
}
int child(int x,int t){pb(son[x][t]);return son[x][t];}
void rotate(int x,int t){
  int y=f[x],w=(son[y][t+1]==x)+t;
  son[y][w]=son[x][w^1];
  if(son[x][w^1])f[son[x][w^1]]=y;
  if(f[y])for(int z=f[y],i=0;i<4;i++)if(son[z][i]==y)son[z][i]=x;
  f[x]=f[y];f[y]=x;son[x][w^1]=y;up(y);
}
void splay(int x,int t=0){
  int s=1,i=x,y;a[1]=i;
  while(!isroot(i,t))a[++s]=i=f[i];
  while(s)pb(a[s--]);
  while(!isroot(x,t)){
    y=f[x];
    if(!isroot(y,t)){if((son[f[y]][t]==y)^(son[y][t]==x))rotate(x,t);else rotate(y,t);}
    rotate(x,t);
  }
  up(x);
}
int newnode(){
  int x=rub?ru[rub--]:++tot;
  son[x][2]=son[x][3]=0;in[x]=1;
  return x;
}
void setson(int x,int t,int y){son[x][t]=y;f[y]=x;}
int pos(int x){for(int i=0;i<4;i++)if(son[f[x]][i]==x)return i;return 4;}
void add(int x,int y){//从x连出一条虚边到y
  if(!y)return;
  pb(x);
  for(int i=2;i<4;i++)if(!son[x][i]){
    setson(x,i,y);
    return;
  }
  while(son[x][2]&&in[son[x][2]])x=child(x,2);
  int z=newnode();
  setson(z,2,son[x][2]);
  setson(z,3,y);
```

```
  setson(x,2,z);
  splay(z,2);
}
void del(int x){//将x与其虚边上的父亲断开
  if(!x)return;
  splay(x);
  if(!f[x])return;
  int y=f[x];
  if(in[y]){
    int s=1,i=y,z=f[y];a[1]=i;
    while(!isroot(i,2))a[++s]=i=f[i];
    while(s)pb(a[s--]);
    if(z){
      setson(z,pos(y),child(y,pos(x)^1));
      splay(z,2);
    }
    ru[++rub]=y;
  }else{
    son[y][pos(x)]=0;
    splay(y);
  }
  f[x]=0;
}
int fa(int x){//x通过虚边的父亲
  splay(x);
  if(!f[x])return 0;
  if(!in[f[x]])return f[x];
  int t=f[x];
  splay(t,2);
  return f[t];
}
int access(int x){
  int y=0;
  for(;x;y=x,x=fa(x)){
    splay(x);
    del(y);
    add(x,son[x][1]);
    setson(x,1,y);
    up(x);
  }
  return y;
}
int lca(int x,int y){
  access(x);
  return access(y);
}
int root(int x){
  access(x);
  splay(x);
  while(son[x][0])x=son[x][0];
  return x;
}
void makeroot(int x){
  access(x);
  splay(x);
  rev1(x);
}
```

```
void link(int x,int y){
  makeroot(x);
  add(y,x);
  access(x);
}
void cut(int x){
  access(x);
  splay(x);
  f[son[x][0]]=0;
  son[x][0]=0;
  up(x);
}
void changechain(int x,int y,tag p){
  makeroot(x);
  access(y);
  splay(y);
  tagchain(y,p);
}
data askchain(int x,int y){
  makeroot(x);
  access(y);
  splay(y);
  return csum[y];
}
void changetree(int x,tag p){
  access(x);
  splay(x);
  val[x]=atag(val[x],p);
  for(int i=2;i<4;i++)if(son[x][i])tagtree(son[x][i],p,1);
  up(x);
  splay(x);
}
data asktree(int x){
  access(x);
  splay(x);
  data t=data(val[x]);
  for(int i=2;i<4;i++)if(son[x][i])t=t+asum[son[x][i]];
  return t;
}
int n,m,x,y,z,k,i,ed[N][2];
int main(){
  read(n);read(m);
  tot=n;
  for(i=1;i<n;i++)read(ed[i][0]),read(ed[i][1]);
  for(i=1;i<=n;i++)read(val[i]),up(i);
  for(i=1;i<n;i++)link(ed[i][0],ed[i][1]);
  read(rt);
  makeroot(rt);
  while(m--){
    read(k);
    if(k==1){//换根
      read(rt);
      makeroot(rt);
    }
    if(k==9){//x的父亲变成y
      read(x),read(y);
      if(lca(x,y)==x)continue;
```

```
      cut(x);
      link(y,x);
      makeroot(rt);
    }
    if(k==0){//子树赋值
      read(x),read(y);
      changetree(x,tag(0,y));
    }
    if(k==5){//子树加
      read(x),read(y);
      changetree(x,tag(1,y));
    }
    if(k==3){//子树最小值
      read(x);
      printf("%d\n",asktree(x).minv);
    }
    if(k==4){//子树最大值
      read(x);
      printf("%d\n",asktree(x).maxv);
    }
    if(k==11){//子树和
      read(x);
      printf("%d\n",asktree(x).sum);
    }
    if(k==2){//链赋值
      read(x),read(y),read(z);
      changechain(x,y,tag(0,z));
      makeroot(rt);
    }
    if(k==6){//链加
      read(x),read(y),read(z);
      changechain(x,y,tag(1,z));
      makeroot(rt);
    }
    if(k==7){//链最小值
      read(x),read(y);
      printf("%d\n",askchain(x,y).minv);
      makeroot(rt);
    }
    if(k==8){//链最大值
      read(x),read(y);
      printf("%d\n",askchain(x,y).maxv);
      makeroot(rt);
    }
    if(k==10){//链和
      read(x),read(y);
      printf("%d\n",askchain(x,y).sum);
      makeroot(rt);
    }
  }
}
```

3.6 Splay

3.6.1 普通 Splay

1.ADD x y D：区间 $[x, y]$ 加 D。

2.REVERSE x y：将区间 $[x, y]$ 翻转。

3.REVOLVE x y T：将区间 $[x, y]$ 向右旋转 T 个单位。

4.INSERT x P：在第 x 个数后插入 P。

5.DELETE x：删去第 x 个数。

6.MIN x y：查询区间 $[x, y]$ 内的最小值。

```
int n,m,a[N],val[N],mn[N],tag[N],size[N],son[N][2],f[N],tot,root;bool rev[N];
void rev1(int x){if(!x)return;swap(son[x][0],son[x][1]);rev[x]^=1;}
void add1(int x,int p){if(!x)return;val[x]+=p;mn[x]+=p;tag[x]+=p;}
void pb(int x){
  if(rev[x]){
    rev1(son[x][0]);
    rev1(son[x][1]);
    rev[x]=0;
  }
  if(tag[x]){
    add1(son[x][0],tag[x]);
    add1(son[x][1],tag[x]);
    tag[x]=0;
  }
}
void up(int x){
  size[x]=1,mn[x]=val[x];
  if(son[x][0]){
    size[x]+=size[son[x][0]];
    if(mn[x]>mn[son[x][0]])mn[x]=mn[son[x][0]];
  }
  if(son[x][1]){
    size[x]+=size[son[x][1]];
    if(mn[x]>mn[son[x][1]])mn[x]=mn[son[x][1]];
  }
}
void rotate(int x){
  int y=f[x],w=son[y][1]==x;
  son[y][w]=son[x][w^1];
  if(son[x][w^1])f[son[x][w^1]]=y;
  if(f[y]){
    int z=f[y];
    if(son[z][0]==y)son[z][0]=x;
    if(son[z][1]==y)son[z][1]=x;
  }
  f[x]=f[y];son[x][w^1]=y;f[y]=x;up(y);
}
void splay(int x,int w){
  int s=1,i=x,y;a[1]=x;
  while(f[i])a[++s]=i=f[i];
  while(s)pb(a[s--]);
  while(f[x]!=w){
    y=f[x];
```

```
    if(f[y]!=w){if((son[f[y]][0]==y)^(son[y][0]==x))rotate(x);else rotate(y);}
    rotate(x);
  }
  if(!w)root=x;
  up(x);
}
int build(int l,int r,int fa){
  int x=++tot,mid=(l+r)>>1;
  f[x]=fa;val[x]=a[mid];
  if(l<mid)son[x][0]=build(l,mid-1,x);
  if(r>mid)son[x][1]=build(mid+1,r,x);
  up(x);
  return x;
}
int kth(int k){
  int x=root,tmp;
  while(1){
    pb(x);
    tmp=size[son[x][0]]+1;
    if(k==tmp)return x;
    if(k<tmp)x=son[x][0];else k-=tmp,x=son[x][1];
  }
}
int main(){
  scanf("%d",&n);
  for(int i=1;i<=n;i++)scanf("%d",&a[i]);
  root=build(0,n+1,0);
  scanf("%d",&m);
  while(m--){
    char op[9];int x,y,z;
    scanf("%s%d",op,&x);
    if(op[0]=='A'){
      scanf("%d%d",&y,&z);
      x=kth(x),y=kth(y+2);
      splay(x,0),splay(y,x),add1(son[y][0],z);
    }
    if(op[0]=='R'&&op[3]=='E'){
      scanf("%d",&y);
      x=kth(x),y=kth(y+2);
      splay(x,0),splay(y,x),rev1(son[y][0]);
    }
    if(op[0]=='R'&&op[3]=='O'){
      scanf("%d%d",&y,&z),z%=y-x+1;
      if(z){
        int u=x,t;
        x=kth(y-z+1),y=kth(y+2);
        splay(x,0),splay(y,x),t=son[y][0];
        son[y][0]=0,up(y),up(x);
        x=kth(u),y=kth(u+1);
        splay(x,0),splay(y,x),son[y][0]=t,f[t]=y;
        up(y),up(x);
      }
    }
    if(op[0]=='I'){
      scanf("%d",&y);
      x=kth(x+1);
      splay(x,0);
```

```
      f[++tot]=x,val[tot]=y;
      son[tot][1]=son[x][1],f[son[x][1]]=tot,son[x][1]=tot;
      up(tot),up(x);
    }
    if(op[0]=='D'){
      y=x;
      x=kth(x),y=kth(y+2);
      splay(x,0),splay(y,x),son[y][0]=0;
      up(y),up(x);
    }
    if(op[0]=='M'){
      scanf("%d",&y);
      x=kth(x),y=kth(y+2);
      splay(x,0),splay(y,x),printf("%d\n",mn[son[y][0]]);
    }
  }
}
```

3.6.2 缩点 Splay

0 p a b：在 p 位置和 $p+1$ 位置之间插入整数 $a, a+1, a+2, ..., b-1, b$。

1 a b：删除 $a, a+1, a+2, ..., b-1, b$ 位置的元素。

2 p：查询 p 位置的元素。

```
int n,m,i,k,x,y,z,tmp[N],tot,root,f[N],son[N][2],l[N],r[N],sum[N];
void build(int fa,int a,int b){
  int mid=(a+b)>>1,x=++tot;
  f[x]=fa,l[x]=r[x]=tmp[mid],sum[x]=b-a+1;
  if(a==b)return;
  if(mid>a)son[x][0]=tot+1,build(x,a,mid-1);
  if(mid<b)son[x][1]=tot+1,build(x,mid+1,b);
}
void up(int x){sum[x]=sum[son[x][0]]+sum[son[x][1]]+r[x]-l[x]+1;}
void setson(int x,int w,int y){son[x][w]=y;if(y)f[y]=x;}
void rotate(int x){
  int y=f[x],w=son[y][1]==x;
  son[y][w]=son[x][!w];
  if(son[x][!w])f[son[x][!w]]=y;
  if(f[y]){
    int z=f[y];
    if(son[z][0]==y)son[z][0]=x;
    if(son[z][1]==y)son[z][1]=x;
  }
  f[x]=f[y];f[y]=x;son[x][!w]=y;up(y);
}
void splay(int x,int w=0){
  while(f[x]!=w){
    int y=f[x];
    if(f[y]!=w){if((son[f[y]][0]==y)^(son[y][0]==x))rotate(x);else rotate(y);}
    rotate(x);
  }
  up(x);
  if(!w)root=x;
}
```

```
int kth(int k,int id=0){
  int x=root,nl,nr;
  while(1){
    nl=sum[son[x][0]]+1;nr=nl+r[x]-l[x];
    if(nl<=k&&k<=nr)return id?x:k-nl+l[x];
    if(k<nl)x=son[x][0];
    else k-=nr,x=son[x][1];
  }
}
int takeout(int k){
  int x=kth(k,1),val=kth(k);
  splay(x);
  int tl=l[x],tr=r[x],sl=son[x][0],sr=son[x][1];
  l[x]=r[x]=val;
  if(val!=tl){
    int y=++tot;
    l[y]=tl;r[y]=val-1;
    setson(y,0,sl);up(y);setson(x,0,y);
  }else setson(x,0,sl);
  if(val!=tr){
    int y=++tot;
    l[y]=val+1;r[y]=tr;
    setson(y,1,sr);up(y);setson(x,1,y);
  }else setson(x,1,sr);
  up(x);
  return x;
}
void ins(int k,int a,int b){
  int y=++tot;
  takeout(k+1);
  l[y]=a,r[y]=b;
  setson(y,1,son[root][1]);up(y);setson(root,1,y);up(root);
}
void del(int a,int b){
  int x=takeout(b+2);
  takeout(a);
  splay(x,root);setson(x,0,0);up(x);up(root);
}
int main(){
  scanf("%d%d",&n,&m);
  for(i=root=1;i<=n;i++)scanf("%d",tmp+i);
  build(0,0,n+1);
  while(m--){
    scanf("%d%d",&k,&x);
    if(k==0)scanf("%d%d",&y,&z),ins(x,y,z);
    if(k==1)scanf("%d",&y),del(x,y);
    if(k==2)printf("%d\n",kth(x+1));
  }
}
```

3.7 Treap

```
struct node{
  int val,cnt,sum,p;node *l,*r;
  node(){val=cnt=sum=p=0;l=r=NULL;}
  void up(){sum=cnt+l->sum+r->sum;}
}*blank=new(node),*root;
void Init(){
  blank->l=blank->r=blank;
  root=blank;
}
void Rotatel(node*&x){node*y=x->r;x->r=y->l;x->up();y->l=x;y->up();x=y;}
void Rotater(node*&x){node*y=x->l;x->l=y->r;x->up();y->r=x;y->up();x=y;}
//插入一个p
void Insert(node*&x,int p){
  if(x==blank){
    x=new(node);x->val=p;x->l=x->r=blank;x->cnt=x->sum=1;x->p=rand();
    return;
  }
  x->sum++;
  if(p==x->val){x->cnt++;return;}
  if(p<x->val){
    Insert(x->l,p);
    if(x->l->p>x->p)Rotater(x);
  }else{
    Insert(x->r,p);
    if(x->r->p>x->p)Rotatel(x);
  }
}
//删除一个p
void Delete(node*&x,int p){
  x->sum--;
  if(p==x->val){x->cnt--;return;}
  if(p<x->val)Delete(x->l,p);else Delete(x->r,p);
}
//查询大于p的数字的个数
int Ask(node*&x,int p){
  if(x==blank)return 0;
  if(p==x->val)return x->r->sum;
  if(p<x->val)return x->cnt+x->r->sum+Ask(x->l,p);
  return Ask(x->r,p);
}
//查询在[c,d]范围内的数字的个数
int Ask(node*&x,int a,int b,int c,int d){
  if(x==blank)return 0;
  if(c<=a&&b<=d)return x->sum;
  int t=c<=x->val&&x->val<=d?x->cnt:0;
  if(c<x->val)t+=Ask(x->l,a,x->val-1,c,d);
  if(d>x->val)t+=Ask(x->r,x->val+1,b,c,d);
}
```

3.8 替罪羊树实现动态标号

维护一个序列，一开始为空，支持以下操作：

1.Insert x：在序列中插入一个数，且插入后它位于从左往右第 x 个。

2.Ask x y：询问第 x 插入的数和第 y 个插入的数中哪一个在左边。

用替罪羊树支持动态标号，对于查询 x,y，等价于比较 $tm[x]$ 与 $tm[y]$ 哪个更小。插入 $O(\log n)$，查询 $O(1)$。

```
#include<cstdio>
#include<cmath>
#define N 400010
using namespace std;
typedef unsigned long long ll;
const ll inf=1ULL<<63;
const double A=0.8;
ll tl[N],tr[N],tm[N];
int size[N],son[N][2],f[N],tot,root;
int id[N],cnt;
int ins(int x,int b){
  size[x]++;
  if(!son[x][b]){
    son[x][b]=++tot;f[tot]=x;size[tot]=1;
    if(!b)tl[tot]=tl[x],tr[tot]=tm[x];else tl[tot]=tm[x],tr[tot]=tr[x];
    tm[tot]=(tl[tot]+tr[tot])>>1;
    return tot;
  }else return ins(son[x][b],b);
}
void dfs(int x){
  if(son[x][0])dfs(son[x][0]);
  id[++cnt]=x;
  if(son[x][1])dfs(son[x][1]);
}
int build(int fa,int l,int r,ll a,ll b){
  int mid=(l+r)>>1,x=id[mid];
  f[x]=fa;son[x][0]=son[x][1]=0;size[x]=1;tl[x]=a;tr[x]=b;tm[x]=(a+b)>>1;
  if(l==r)return x;
  if(l<mid)size[x]+=size[son[x][0]=build(x,l,mid-1,a,tm[x])];
  if(r>mid)size[x]+=size[son[x][1]=build(x,mid+1,r,tm[x],b)];
  return x;
}
int rebuild(int x){
  cnt=0;dfs(x);return build(f[x],1,cnt,tl[x],tr[x]);
}
int kth(int k){
  int x=root,rank;
  while(1){
    size[x]++;
    rank=size[son[x][0]]+1;
    if(k==rank)return x;
    if(k<rank)x=son[x][0];else k-=rank,x=son[x][1];
  }
}
void kthins(int k){
  if(!root){root=tot=size[1]=1;tr[1]=inf,tm[1]=inf>>1;return;}
  int x;
```

```
  if(k==1)x=ins(root,0);
  else if(k>size[root])x=ins(root,1);
  else{
    x=kth(k);
    if(son[x][0])x=ins(son[x][0],1);else{
      son[x][0]=++tot;f[tot]=x;size[tot]=1;
      tl[tot]=tl[x],tr[tot]=tm[x];
      tm[tot]=(tl[tot]+tr[tot])>>1;
      x=tot;
    }
  }
  int deep=1;int z=x;while(f[z])z=f[z],deep++;
  if(deep<log(tot)/log(1/A))return;
  while((double)size[son[x][0]]<A*size[x]&&(double)size[son[x][1]]<A*size[x])x=f[x];
  if(!x)return;
  if(x==root){root=rebuild(x);return;}
  int y=f[x],b=son[y][1]==x,now=rebuild(x);
  son[y][b]=now;
}
```

3.9 权值线段树中位数查询

离散化后一共有 m 个元素，支持以下操作：

1.ins c d e：插入一个离散化后是 c 的元素，对 cnt 的贡献为 d，对 sum 的贡献为 e。

2.ask：查询所有数字与中位数的差值的绝对值的和。

```
struct T{
int v[N];ll sum[N];
void ins(int c,int d,int e){
  int a=1,b=m,x=1,mid;
  while(1){
    v[x]+=d,sum[x]+=e;
    if(a==b)return;
    x<<=1;
    if(c<=(mid=(a+b)>>1))b=mid;else x|=1,a=mid+1;
  }
}
ll ask(){
  if(v[1]<=1)return 0;
  int a=1,b=m,mid,t,k=(v[1]+1)/2,x=1,cnt=0;ll ans=0;
  while(a<b){
    mid=(a+b)>>1,t=v[x<<=1];
    if(k<=t)cnt+=v[x|1],ans+=sum[x|1],b=mid;
    else cnt-=t,ans-=sum[x],k-=t,a=mid+1,x|=1;
  }
  return ans-sum[x]/v[x]*cnt;
}
};
```

3.10 线段树合并

合并根为 x, y 的两棵线段树，区间范围为 $[a, b]$。

```
int merge(int x,int y,int a,int b){
  if(!x)return y;
  if(!y)return x;
  int z=++tot;
  if(a==b){
    v[z]=v[x]+v[y];
    return z;
  }
  int mid=(a+b)>>1;
  l[z]=merge(l[x],l[y],a,mid);
  r[z]=merge(r[x],r[y],mid+1,b);
  v[z]=v[l[z]]+v[r[z]];
  return z;
}
```

3.11 树链剖分

```
void dfs(int x){
  size[x]=1;
  for(int i=g[x];i;i=nxt[i])if(v[i]!=f[x]){
    f[v[i]]=x,d[v[i]]=d[x]+1;
    dfs(v[i]),size[x]+=size[v[i]];
    if(size[v[i]]>size[son[x]])son[x]=v[i];
  }
}
void dfs2(int x,int y){
  st[x]=++dfn;top[x]=y;
  if(son[x])dfs2(son[x],y);
  for(int i=g[x];i;i=nxt[i])if(v[i]!=son[x]&&v[i]!=f[x])dfs2(v[i],v[i]);
  en[x]=dfn;
}
//查询x,y两点的lca
int lca(int x,int y){
  for(;top[x]!=top[y];x=f[top[x]])if(d[top[x]]<d[top[y]]){int z=x;x=y;y=z;}
  return d[x]<d[y]?x:y;
}
//x是y的祖先，查询x到y方向的第一个点
int lca2(int x,int y){
  int t;
  while(top[x]!=top[y])t=top[y],y=f[top[y]];
  return x==y?t:son[x];
}
//以root为根对x的子树操作
void subtree(int x){
  if(x==root){change(1,n);return;}
  if(st[x]>st[root]||en[x]<en[root]){change(st[x],en[x]);return;}
  int y=lca2(root,x);
  change(st[y]-1,p),change(en[y]+1,n);
}
//对x到y路径上的点进行操作
void chain(int x,int y){
```

```
  for(;top[x]!=top[y];x=f[top[x]]){
    if(d[top[x]]<d[top[y]]){int z=x;x=y;y=z;}
    change(st[top[x]],st[x]);
  }
  if(d[x]<d[y]){int z=x;x=y;y=z;}
  change(st[y],st[x]);
}
```

3.12 李超线段树

支持插入一条线段，查询横坐标为某个值时最上面的线段。插入 $O(\log^2 n)$，查询 $O(\log n)$。

```
#include<cstdio>
#include<cmath>
#include<algorithm>
#define N 39989
using namespace std;
struct Seg{
  double k,b;
  Seg(){}
  Seg(int x0,int y0,int x1,int y1){
    if(x0==x1)k=0,b=max(y0,y1);
    else k=1.0*(y0-y1)/(x0-x1),b=-k*x0+y0;
  }
  double gety(int x){return k*x+b;}
}s[100010];
int m,op,cnt,X0,Y0,X1,Y1,ans,v[131000];
inline int sig(double x){return fabs(x)<1e-8?0:(x>0?1:-1);}
void ins(int x,int a,int b,int c,int d,int p){
  if(c<=a&&b<=d){
    if(sig(s[p].gety(a)-s[v[x]].gety(a))>0
       &&sig(s[p].gety(b)-s[v[x]].gety(b))>0){v[x]=p;return;}
    if(sig(s[p].gety(a)-s[v[x]].gety(a))<=0
       &&sig(s[p].gety(b)-s[v[x]].gety(b))<=0)return;
    if(a==b)return;
  }
  int mid=(a+b)>>1;
  if(c<=mid)ins(x<<1,a,mid,c,d,p);
  if(d>mid)ins(x<<1|1,mid+1,b,c,d,p);
}
void ask(int x,int a,int b,int c){
  if(sig(s[ans].gety(c)-s[v[x]].gety(c))<0)ans=v[x];
  else if(!sig(s[ans].gety(c)-s[v[x]].gety(c))&&ans>v[x])ans=v[x];
  if(a==b)return;
  int mid=(a+b)>>1;
  c<=mid?ask(x<<1,a,mid,c):ask(x<<1|1,mid+1,b,c);
}
int main(){
  s[0].b=-1;
  read(m);
  while(m--){
    read(op);
    if(!op){
      read(X0);
      ans=0,ask(1,1,N,X0);
```

```
      printf("%d\n",ans);
    }else{
      read(X0),read(Y0),read(X1),read(Y1);
      s[++cnt]=Seg(X0,Y0,X1,Y1);
      ins(1,1,N,X0,X1,cnt);
    }
  }
}
```

3.13 ST 表

```
int Log[N],f[17][N];
int ask(int x,int y){
  int k=log[y-x+1];
  return max(f[k][x],f[k][y-(1<<k)+1]);
}
int main(){
  for(i=2;i<=n;i++)Log[i]=Log[i>>1]+1;
  for(j=1;j<K;j++)for(i=1;i+(1<<j-1)<=n;i++)f[j][i]=max(f[j-1][i],f[j-1][i+(1<<j-1)]);
}
```

3.14 左偏树

顶部为最小元素。

```
typedef pair<int,int>P;
struct Node{
  int l,r,d;P v;
  Node(){}
  Node(int _l,int _r,int _d,P _v){l=_l,r=_r,d=_d,v=_v;}
}T[N];
int merge(int a,int b){
  if(!a)return b;
  if(!b)return a;
  if(T[a].v>T[b].v)swap(a,b);
  T[a].r=merge(T[a].r,b);
  if(T[T[a].l].d<T[T[a].r].d)swap(T[a].l,T[a].r);
  T[a].d=T[a].r?T[T[a].r].d+1:0;
  return a;
}
int pop(int a){
  int l=T[a].l,r=T[a].r;
  T[a].l=T[a].r=T[a].d=0;
  return merge(l,r);
}
```

3.15 带修改区间第 k 小

维护一个序列 a，支持以下操作：

1 x y：把 $a[x]$ 修改为 y。

2 x y k：查询 $[x,y]$ 内第 k 小值。

时间复杂度 $O(n\log^2 n)$，空间复杂度 $O(n\log n)$。

```
#include<cstdio>
#include<cstdlib>
#include<algorithm>
using namespace std;
const int N=200010,M=524289;
int n,m,cnt,i,a[N],op[N][4],b[N];
int lower(int x){
  int l=1,r=cnt,t,mid;
  while(l<=r)if(b[mid=(l+r)>>1]<=x)l=(t=mid)+1;else r=mid-1;
  return t;
}
struct node{
  int val,cnt,sum,p;node*l,*r;
  node(){val=sum=cnt=p=0;l=r=NULL;}
  inline void up(){sum=l->sum+r->sum+cnt;}
}*blank=new(node),*T[M],pool[2000000],*cur;
void Rotatel(node*&x){node*y=x->r;x->r=y->l;x->up();y->l=x;y->up();x=y;}
void Rotater(node*&x){node*y=x->l;x->l=y->r;x->up();y->r=x;y->up();x=y;}
void Ins(node*&x,int y,int p){
  if(x==blank){x=cur++;x->val=y;x->l=x->r=blank;x->sum=x->cnt=1;x->p=std::rand();return;}
  x->sum+=p;
  if(y==x->val){x->cnt+=p;return;}
  if(y<x->val){
    Ins(x->l,y,p);
    if(x->l->p>x->p)Rotater(x);
  }else{
    Ins(x->r,y,p);
    if(x->r->p>x->p)Rotatel(x);
  }
}
int Ask(node*x,int y){//ask how many <= y
  int t=0;
  while(x!=blank)if(y<x->val)x=x->l;else t+=x->l->sum+x->cnt,x=x->r;
  return t;
}
void add(int v,int i,int p){
  int a=1,b=cnt,mid,f=1,x=1;
  while(a<b){
    if(f)Ins(T[x],i,p);
    mid=(a+b)>>1;x<<=1;
    if(f=v<=mid)b=mid;else a=mid+1,x|=1;
  }
  Ins(T[x],i,p);
}
int kth(int l,int r,int k){
  int x=1,a=1,b=cnt,mid;
  while(a<b){
    mid=(a+b)>>1;x<<=1;
    int t=Ask(T[x],r)-Ask(T[x],l-1);
    if(k<=t)b=mid;else k-=t,a=mid+1,x|=1;
  }
  return a;
}
void build(int x,int a,int b){
  T[x]=blank;
```

```
  if(a==b)return;
  int mid=(a+b)>>1;
  build(x<<1,a,mid),build(x<<1|1,mid+1,b);
}
int main(){
  blank->l=blank->r=blank;
  while(~scanf("%d",&n)){
    cur=pool;
    for(i=1;i<=n;i++)read(a[i]),b[i]=a[i];
    cnt=n;
    read(m);
    for(i=1;i<=m;i++){
      read(op[i][0]),read(op[i][1]),read(op[i][2]);
      if(op[i][0]==1)b[++cnt]=op[i][2];else read(op[i][3]);
    }
    sort(b+1,b+cnt+1);
    for(i=1;i<=n;i++)a[i]=lower(a[i]);
    for(i=1;i<=m;i++)if(op[i][0]==1)op[i][2]=lower(op[i][2]);
    build(1,1,cnt);
    for(i=1;i<=n;i++)add(a[i],i,1);
    for(i=1;i<=m;i++){
      if(op[i][0]==1)add(a[op[i][1]],op[i][1],-1),add(a[op[i][1]]=op[i][2],op[i][1],1);
      else printf("%d\n",b[kth(op[i][1],op[i][2],op[i][3])]);
    }
  }
}
```

4 树

4.1 动态维护树的带权重心

支持单点修改，查询带权重心到所有点的带权距离和。修改 $O(\log^2 n)$，查询 $O(\log n)$。

```
#include<cstdio>
typedef long long ll;
const int N=100010,M=2000000,T=262145;
int n,m,i,x,y,z;
int g[N],nxt[N<<1],v[N<<1],w[N<<1],ok[N<<1],ed,son[N],f[N],all,now,cnt,value[N];
int size[N],heavy[N],top[N],loc[N],seq[N],dfn;
int G[N],NXT[M],V[2][M],W[M],ED,tag[T];
ll val[T],sw[N],sdw[N],sew[N],sedw[N];
void add(int x,int y,int z){v[++ed]=y,w[ed]=z,nxt[ed]=g[x],ok[ed]=1,g[x]=ed;}
void ADD(int x,int y,int z,int w){V[0][++ED]=y;V[1][ED]=z;W[ED]=w;NXT[ED]=G[x];G[x]=ED;}
void findroot(int x,int pre){
  son[x]=1;f[x]=0;
  for(int i=g[x];i;i=nxt[i])if(ok[i]&&v[i]!=pre){
    findroot(v[i],x);
    son[x]+=son[v[i]];
    if(son[v[i]]>f[x])f[x]=son[v[i]];
  }
  if(all-son[x]>f[x])f[x]=all-son[x];
  if(f[x]<f[now])now=x;
}
void dfs(int x,int pre,int dis){
  ADD(x,now,cnt,dis);
  for(int i=g[x];i;i=nxt[i])if(ok[i]&&v[i]!=pre)dfs(v[i],x,dis+w[i]);
}
void solve(int x){
  int i;
  for(i=g[x];i;i=nxt[i])if(ok[i])++cnt,dfs(v[i],x,w[i]);
  for(i=g[x];i;i=nxt[i])if(ok[i]){
    ok[i^1]=0;
    f[0]=all=son[v[i]];
    findroot(v[i],now=0);
    solve(now);
  }
}
void dfs1(int x,int y){
  size[x]=1;f[x]=y;
  for(int i=g[x];i;i=nxt[i])if(v[i]!=y){
    dfs1(v[i],x);size[x]+=size[v[i]];
    if(size[v[i]]>size[heavy[x]])heavy[x]=v[i];
  }
}
void dfs2(int x,int y){
  top[x]=y;seq[loc[x]=++dfn]=x;
  if(heavy[x])dfs2(heavy[x],y);
  for(int i=g[x];i;i=nxt[i])if(v[i]!=heavy[x]&&v[i]!=f[x])dfs2(v[i],v[i]);
}
void add1(int x,int p){val[x]+=p,tag[x]+=p;}
void pb(int x){if(tag[x])add1(x<<1,tag[x]),add1(x<<1|1,tag[x]),tag[x]=0;}
void change(int x,int a,int b,int c,int d,int p){
  if(c<=a&&b<=d){add1(x,p);return;}
```

```
  pb(x);
  int mid=(a+b)>>1;
  if(c<=mid)change(x<<1,a,mid,c,d,p);
  if(d>mid)change(x<<1|1,mid+1,b,c,d,p);
  val[x]=val[x<<1]>val[x<<1|1]?val[x<<1]:val[x<<1|1];
}
int getroot(){
  int x=1,a=1,b=n,mid;
  while(a<b){
    pb(x),mid=(a+b)>>1;
    if(val[x<<1|1]*2>=val[1])a=mid+1,x=x<<1|1;else b=mid,x<<=1;
  }
  return seq[a];
}
void modify(int x,int y){
  value[x]+=y;
  for(int i=G[x];i;i=NXT[i]){
    sw[V[0][i]]+=y,sdw[V[0][i]]+=(ll)W[i]*y;
    sew[V[1][i]]+=y,sedw[V[1][i]]+=(ll)W[i]*y;
  }
  while(top[x]!=1)change(1,1,n,loc[top[x]],loc[x],y),x=f[top[x]];
  change(1,1,n,1,loc[x],y);
}
ll query(int x){
  ll t=sdw[x];
  for(int i=G[x];i;i=NXT[i])
    t+=(sw[V[0][i]]-sew[V[1][i]]+value[V[0][i]])*W[i]+sdw[V[0][i]]-sedw[V[1][i]];
  return t;
}
int main(){
  read(n),read(m);
  for(ed=i=1;i<n;i++)read(x),read(y),read(z),add(x,y,z),add(y,x,z);
  f[0]=all=n;findroot(1,now=0);solve(now);
  dfs1(1,0),dfs2(1,1);
  while(m--)read(x),read(y),modify(x,y),printf("%lld\n",query(getroot()));
}
```

4.2 支持加边的树的重心的维护

ans 表示每个连通块的重心到其它点的距离和的和，时间复杂度 $O(n\log^2 n)$。

```
int n,m,i,x,y,ans;char op[5];
int g[N],v[N<<1],nxt[N<<1],; ed
int f[N],son[N][2],val[N],tag[N],sum[N],ts[N],td[N],size[N],tmp[N];
bool isroot(int x){return !f[x]||son[f[x]][0]!=x&&son[f[x]][1]!=x;}
void add1(int x,int p){if(!x)return;val[x]+=p;tag[x]+=p;}
void add2(int x,int s,int d){if(!x)return;sum[x]+=s+size[son[x][1]]*d;ts[x]+=s;td[x]+=d;}
void pb(int x){
  if(tag[x]){
    add1(son[x][0],tag[x]);
    add1(son[x][1],tag[x]);
    tag[x]=0;
  }
  if(td[x]){
    add2(son[x][0],ts[x]+(size[son[x][1]]+1)*td[x],td[x]);
```

```
    add2(son[x][1],ts[x],td[x]);
    ts[x]=td[x]=0;
  }
}
void up(int x){size[x]=size[son[x][0]]+size[son[x][1]]+1;}
void rotate(int x){
  int y=f[x],w=son[y][1]==x;
  son[y][w]=son[x][w^1];
  if(son[x][w^1])f[son[x][w^1]]=y;
  if(f[y]){
    int z=f[y];
    if(son[z][0]==y)son[z][0]=x;else if(son[z][1]==y)son[z][1]=x;
  }
  f[x]=f[y];f[y]=x;son[x][w^1]=y;up(y);
}
void splay(int x){
  int s=1,i=x,y;tmp[1]=i;
  while(!isroot(i))tmp[++s]=i=f[i];
  while(s)pb(tmp[s--]);
  while(!isroot(x)){
    y=f[x];
    if(!isroot(y)){if((son[f[y]][0]==y)^(son[y][0]==x))rotate(x);else rotate(y);}
    rotate(x);
  }
  up(x);
}
void access(int x){for(int y=0;x;y=x,x=f[x])splay(x),son[x][1]=y,up(x);}
int root(int x){access(x);splay(x);while(son[x][0])x=son[x][0];return x;}
void addleaf(int x,int y){
  f[y]=x,son[y][0]=son[y][1]=val[y]=tag[y]=sum[y]=ts[y]=td[y]=0,size[y]=1;
  x=root(x),access(y),splay(x),add1(x,1),add2(x,0,1);
  for(y=son[x][1];son[y][0];y=son[y][0]);splay(y);
  int vx=val[x],vy=val[y];
  if(vy*2>vx){
    val[y]=vx,val[x]-=vy;
    sum[x]-=sum[y]+vy,sum[y]+=sum[x]+vx-vy;
    access(y),splay(x),son[x][0]=y,son[x][1]=0;
  }
}
void dfs(int x,int y){
  addleaf(y,x);
  for(int i=g[x];i;i=nxt[i])if(v[i]!=y)dfs(v[i],x);
}
void addedge(int x,int y){v[++ed]=y;nxt[ed]=g[x];g[x]=ed;}
void link(int x,int y){
  int X=root(x),Y=root(y);
  ans-=sum[X]+sum[Y];
  if(val[X]<val[Y])swap(x,y);
  dfs(y,x),addedge(x,y),addedge(y,x);
  ans+=sum[root(x)];
}
int main(){
  scanf("%d%d",&n,&m);
  for(i=1;i<=n;i++)val[i]=size[i]=1;
  while(m--){
    scanf("%s",op);
    if(op[0]=='A')scanf("%d%d",&x,&y),link(x,y);
```

```
    if(op[0]=='Q')printf("%d\n",ans);
  }
}
```

4.3 虚树

除了 1 之外再给定 m 个点，构造它们的虚树，时间复杂度 $O(m\log n)$。

```
int m,i,a[N],q[N],t,tot;bool vip[N],vis[N];
bool cmp(int x,int y){return st[x]<st[y];}
int main(){
  while(~scanf("%d",&m)){
    vis[1]=vip[1]=a[1]=1;
    for(tot=++m,i=2;i<=m;i++)read(a[i]),vis[a[i]]=vip[a[i]]=1;
    sort(a+1,a+m+1,cmp);
    for(i=1;i<m;i++)if(!vis[x=lca(a[i],a[i+1])])vis[a[++tot]=x]=1;
    m=tot,sort(a+1,a+m+1,cmp);
    for(q[t=1]=1,i=2;i<=m;q[++t]=a[i++]){
      while(st[a[i]]<st[q[t]]||en[a[i]]>en[q[t]])t--;
      addedge(q[t],a[i]);
    }
    for(i=1;i<=m;i++)vis[a[i]]=vip[a[i]]=0;
  }
}
```

4.4 曼哈顿最小生成树

```
int n,m,i,j,w[N],c[N],bit[N],f[N];ll ans;
struct P{
  int x,y,p;
  P(){}
  P(int _x,int _y,int _p){x=_x,y=_y,p=_p;}
}a[N],b[N],e[N<<2];
bool cmp(const P&a,const P&b){return a.x==b.x?a.y<b.y:a.x<b.x;}
bool cmpe(const P&a,const P&b){return a.p<b.p;}
int lower(int x){
  int l=1,r=n,t,mid;
  while(l<=r)if(c[mid=(l+r)>>1]<=x)l=(t=mid)+1;else r=mid-1;
  return t;
}
int abs(int x){return x>0?x:-x;}
int dis(int x,int y){return abs(a[x].x-a[y].x)+abs(a[x].y-a[y].y);}
void ins(int x,int p){for(;x<=n;x+=x&-x)if(w[p]<=w[bit[x]])bit[x]=p;}
int ask(int x){int t=0;for(;x;x-=x&-x)if(w[bit[x]]<=w[t])t=bit[x];return t;}
int F(int x){return f[x]==x?x:f[x]=F(f[x]);}
void solve(){
  for(sort(b+1,b+n+1,cmp),sort(c+1,c+n+1),i=1;i<=n;i++){
    if(j=ask(lower(b[i].y)))e[++m]=P(b[i].p,j,dis(b[i].p,j));
    ins(lower(b[i].y),b[i].p);
  }
}
ll ManhattanMst(){
  for(w[0]=~0U>>1,m=ans=0,i=1;i<=n;i++)f[i]=i;
```

```
  for(i=1;i<=n;i++){
    b[i]=P(-a[i].x,a[i].x-a[i].y,i);
    c[i]=b[i].y;
    w[i]=a[i].x+a[i].y;
    bit[i]=0;
  }
  solve();
  for(i=1;i<=n;i++){
    b[i]=P(-a[i].y,a[i].y-a[i].x,i);
    c[i]=b[i].y;
    bit[i]=0;
  }
  solve();
  for(i=1;i<=n;i++){
    b[i]=P(a[i].y,-a[i].x-a[i].y,i);
    c[i]=b[i].y;
    w[i]=a[i].x-a[i].y;
    bit[i]=0;
  |
  solve();
  for(i=1;i<=n;i++){
    b[i]=P(-a[i].x,a[i].y+a[i].x,i);
    c[i]=b[i].y;
    bit[i]=0;
  }
  solve();
  sort(e+1,e+m+1,cmpe);
  for(ans=0,i=1;i<=m;i++)if(F(e[i].x)!=F(e[i].y)){
    f[f[e[i].x]]=f[e[i].y];
    ans+=e[i].p;
  }
  return ans;
}
int main(){
  scanf(”%d”,&n);
  for(i=1;i<=n;i++)scanf(”%d%d”,&a[i].x,&a[i].y);
  printf(”%lld”,ManhattanMst());
}
```

5 图

5.1 欧拉回路

欧拉回路：

无向图：每个顶点的度数都是偶数，则存在欧拉回路。

有向图：每个顶点的入度 = 出度，则存在欧拉回路。

欧拉路径：

无向图：当且仅当该图所有顶点的度数为偶数，或者除了两个度数为奇数外其余的全是偶数。

有向图：当且仅当该图所有顶点出度 = 入度或者一个顶点出度 = 入度 +1，另一个顶点入度 = 出度 +1，其他顶点出度 = 入度。

下面 $O(n+m)$ 求欧拉回路的代码中，n 为点数，m 为边数，若有解则依次输出经过的边的编号，若是无向图，则正数表示 x 到 y，负数表示 y 到 x。

```
namespace UndirectedGraph{
int n,m,i,x,y,d[N],g[N],v[M<<1],w[M<<1],vis[M<<1],nxt[M<<1],ed;
int ans[M],cnt;
void add(int x,int y,int z){
  d[x]++;
  v[++ed]=y;w[ed]=z;nxt[ed]=g[x];g[x]=ed;
}
void dfs(int x){
  for(int&i=g[x];i;){
    if(vis[i]){i=nxt[i];continue;}
    vis[i]=vis[i^1]=1;
    int j=w[i];
    dfs(v[i]);
    ans[++cnt]=j;
  }
}
void solve(){
  scanf(”%d%d”,&n,&m);
  for(i=ed=1;i<=m;i++)scanf(”%d%d”,&x,&y),add(x,y,i),add(y,x,−i);
  for(i=1;i<=n;i++)if(d[i]&1){puts(”NO”);return;}
  for(i=1;i<=n;i++)if(g[i]){dfs(i);break;}
  for(i=1;i<=n;i++)if(g[i]){puts(”NO”);return;}
  puts(”YES”);
  for(i=m;i;i−−)printf(”%d ”,ans[i]);
}
}
namespace DirectedGraph{
int n,m,i,x,y,d[N],g[N],v[M],vis[M],nxt[M],ed;
int ans[M],cnt;
void add(int x,int y){
  d[x]++;d[y]−−;
  v[++ed]=y;nxt[ed]=g[x];g[x]=ed;
}
void dfs(int x){
  for(int&i=g[x];i;){
    if(vis[i]){i=nxt[i];continue;}
    vis[i]=1;
```

```
      int j=i;
      dfs(v[i]);
      ans[++cnt]=j;
    }
}
void solve(){
  scanf("%d%d",&n,&m);
  for(i=1;i<=m;i++)scanf("%d%d",&x,&y),add(x,y);
  for(i=1;i<=n;i++)if(d[i]){puts("NO");return;}
  for(i=1;i<=n;i++)if(g[i]){dfs(i);break;}
  for(i=1;i<=n;i++)if(g[i]){puts("NO");return;}
  puts("YES");
  for(i=m;i;i--)printf("%d ",ans[i]);
}
}
```

5.2 最短路

5.2.1 Dijkstra

```
typedef pair<int,int> P;
priority_queue<P,vector<P>,greater<P> >Q;
void dijkstra(int S){
  int i,x;
  for(i=1;i<=n;i++)d[i]=inf;Q.push(P(d[S]=0,S));
  while(!Q.empty()){
    P t=Q.top();Q.pop();
    if(d[x=t.second]<t.first)continue;
    for(i=g[x];i;i=nxt[i])if(d[x]+w[i]<d[v[i]])Q.push(P(d[v[i]]=d[x]+w[i],v[i]));
  }
}
```

5.2.2 SPFA

```
int q[66000];unsigned short h,t;
void add(int x,int y){
  if(y>=d[x])return;d[x]=y;
  if(!in[x]){
    in[x]=1;
    if(y<d[q[h]])q[--h]=x;else q[++t]=x;//SLF优化
  }
}
void spfa(int S){//S为源点
  int i,x;
  for(i=h=1;i<=n;i++)d[i]=inf,in[i]=0;add(S,t=0);
  while(h!=t+1)for(i=g[x=q[h++]],in[x]=0;i;i=nxt[i])add(v[i],d[x]+w[i]);
}
```

5.2.3 Astar 求 k 短路

求有向图中 S 到 T 的前 k 短路，其中 g 为反图，h 为正图。

```
typedef pair<int,int> P;
const int N=1010,M=10010,inf=1000000010;
int n,m,k,S,T,i,x,y,z;
int g[N],h[N],v[M<<1],w[M<<1],nxt[M<<1],ed,d[N],vis[N],ans[N];
priority_queue<P,vector<P>,greater<P> >Q;
void add(int x,int y,int z){
  v[++ed]=x;w[ed]=z;nxt[ed]=g[y];g[y]=ed;
  v[++ed]=y;w[ed]=z;nxt[ed]=h[x];h[x]=ed;
}
int main(){
  scanf("%d%d%d",&n,&m,&k);S=n,T=1;
  for(i=1;i<=k;i++)ans[i]=-1;
  while(m--)scanf("%d%d%d",&x,&y,&z),add(x,y,z);
  for(i=1;i<=n;i++)d[i]=inf;Q.push(P(d[T]=0,T));
  while(!Q.empty()){
    P t=Q.top();Q.pop();
    if(d[t.second]<t.first)continue;
    for(i=g[x=t.second];i;i=nxt[i])if(d[x]+w[i]<d[v[i]])
      Q.push(P(d[v[i]]=d[x]+w[i],v[i]));
  }
  if(d[S]<inf)Q.push(P(d[S],S));
  while(!Q.empty()){
    P t=Q.top();Q.pop();vis[x=t.second]++;
    if(x==T&&vis[T]<=k)ans[vis[T]]=t.first;
    if(vis[x]<=k)for(i=h[x];i;i=nxt[i])Q.push(P(t.first-d[x]+d[v[i]]+w[i],v[i]));
  }
  for(i=1;i<=k;i++)printf("%d\n",ans[i]);
}
```

5.3 Tarjan

5.3.1 边双连通分量

$cut[i]$ 表示输入的第 i 条边是否是桥边，cnt 表示边双连通分量的个数，$from[i]$ 表示 i 点所属的边双连通分量。

```
int e[M][2],cut[M],g[N],v[M<<1],nxt[M<<1],ed=1;
int f[N],dfn[N],low[N],num,cnt,from[N];
void add(int x,int y){v[++ed]=y;nxt[ed]=g[x];g[x]=ed;}
void tarjan(int x){
  dfn[x]=low[x]=++num;
  for(int i=g[x];i;i=nxt[i])if(!dfn[v[i]]){
    f[v[i]]=i>>1,tarjan(v[i]);
    if(low[x]>low[v[i]])low[x]=low[v[i]];
  }else if(f[x]!=(i>>1)&&low[x]>dfn[v[i]])low[x]=dfn[v[i]];
  if(f[x]&&low[x]==dfn[x])cut[f[x]]=1;
}
void dfs(int x,int y){
  from[x]=y;
  for(int i=g[x];i;i=nxt[i])if(!from[v[i]]&&!cut[i>>1])dfs(v[i],y);
}
int main(){
  scanf("%d%d",&n,&m);
  for(ed=i=1;i<=m;i++){
```

```
    scanf("%d%d",&x,&y);
    e[i][0]=x,e[i][1]=y;
    add(x,y),add(y,x);
  }
  tarjan(1);
  for(i=1;i<=n;i++)if(!from[i])dfs(i,++cnt);
}
```

5.3.2 点双连通分量

```
int dfn[N],low[N],num,cut[N],q[N],t;
void tarjan(int x){
  dfn[x]=low[x]=++num,q[++t]=x;
  for(int i=g[x];i;i=nxt[i])if(!dfn[v[i]]){
    int y=v[i];
    tarjan(y);
    if(low[x]>low[y])low[x]=low[y];
    if(dfn[x]<=low[y]){//x是割点，接下来一行输出所有该点双连通分量内的点
      cut[x]=1;
      while(q[t]!=x)printf("%d ",q[t--]);
      printf("%d\n",x);
    }
  }else if(low[x]>dfn[v[i]])low[x]=dfn[v[i]];
}
```

5.3.3 Dominator Tree

在保证 S 能到达所有点的情况下，求以 S 为源点的 Dominator Tree。

$dfn[x]$：x 的 DFS 序。

$id[x]$：DFS 序第 x 个是什么。

$gd[x]$：DFS 序第 x 个在 Dominator Tree 上的孩子列表。

$idom[x]$：DFS 序第 x 个在 Dominator Tree 上的父亲。

$sd[x]$：DFS 序第 x 个的半必经点。

$id[idom[dfn[x]]]$：x 的最近必经点。

```
#include<cstdio>
const int N=5010,M=200010;//点数，边数
int n,m,i,x,y,q[N],ans;
int g1[N],g2[N],gd[N],v[M*3+N],nxt[M*3+N],ed;
int cnt,dfn[N],id[N],fa[N],f[N],mn[N],sd[N],idom[N];
void add(int*g,int x,int y){v[++ed]=y;nxt[ed]=g[x];g[x]=ed;}
int F(int x){
  if(f[x]==x)return x;
  int y=F(f[x]);
  if(sd[mn[x]]>sd[mn[f[x]]])mn[x]=mn[f[x]];
  return f[x]=y;
}
void dfs(int x){
  id[dfn[x]=++cnt]=x;
  for(int i=g1[x];i;i=nxt[i])if(!dfn[v[i]])dfs(v[i]),fa[dfn[v[i]]]=dfn[x];
}
```

```
void tarjan(int S){
  int i,j,k,x;
  for(cnt=0,i=1;i<=n;i++)gd[i]=dfn[i]=id[i]=fa[i]=idom[i]=0,f[i]=sd[i]=mn[i]=i;
  dfs(S);
  for(i=n;i>1;i--){
    for(j=g2[id[i]];j;j=nxt[j])F(k=dfn[v[j]]),sd[i]=sd[i]<sd[mn[k]]?sd[i]:sd[mn[k]];
    add(gd,sd[i],i);
    for(j=gd[f[i]=x=fa[i]];j;j=nxt[j])F(k=v[j]),idom[k]=sd[mn[k]]<x?mn[k]:x;
    gd[x]=0;
  }
  for(i=2;i<=n;add(gd,idom[i],i),i++)if(idom[i]!=sd[i])idom[i]=idom[idom[i]];
}
int main(){
  while(~scanf(”%d%d”,&n,&m)){
    for(ed=0,i=1;i<=n;i++)g1[i]=g2[i]=0;
    while(m--)scanf(”%d%d”,&x,&y),add(g1,x,y),add(g2,y,x);
    tarjan(1);
  }
}
```

5.4 强连通分量

$G[0]$ 为正图，$G[1]$ 为反图，$G[2]$ 为缩点后的图，$f[i]$ 为 i 所在的 SCC。

```
int G[3][N],NXT[3][M<<1],V[3][M<<1],ed,f[N],q[N],t,vis[N];
void add(int x,int y){
  V[0][++ed]=y;NXT[0][ed]=G[0][x];G[0][x]=ed;
  V[1][ed]=x;NXT[1][ed]=G[1][y];G[1][y]=ed;
}
void ADD(int x,int y){V[2][++ed]=y;NXT[2][ed]=G[2][x];G[2][x]=ed;}
void dfs1(int x){
  vis[x]=1;
  for(int i=G[0][x];i;i=NXT[0][i])if(!vis[V[0][i]])dfs1(V[0][i]);
  q[++t]=x;
}
void dfs2(int x,int y){
  vis[x]=0,f[x]=y;
  for(int i=G[1][x];i;i=NXT[1][i])if(vis[V[1][i]])dfs2(V[1][i],y);
}
int main(){
  for(t=0,i=1;i<=n;i++)if(!vis[i])dfs1(i);
  for(i=n;i;i--)if(vis[q[i]])dfs2(q[i],q[i]);
  for(ed=0,i=1;i<=n;i++)for(j=G[0][i];j;j=NXT[0][j])
    if(f[i]!=f[V[0][j]])ADD(f[i],f[V[0][j]]);
}
```

5.5 无负权图的最小环

有向图：$d[i][i]=inf$，然后跑 floyd，$ans=\min(d[i][i])$。

求无向图中经过至少 3 个点的最小环代码如下：

```
int main(){
  for(i=1;i<=n;i++)for(j=1;j<=n;j++)g[i][j]=d[i][j]=inf;
```

```
  while(m--){
    scanf("%d%d%d",&x,&y,&z);
    if(z<g[x][y])g[x][y]=g[y][x]=d[x][y]=d[y][x]=z;
  }
  for(ans=inf,k=1;k<=n;k++){
    for(i=1;i<k;i++)for(j=i+1;j<k;j++)ans=min(ans,d[i][j]+g[i][k]+g[k][j]);
    for(i=1;i<=n;i++)for(j=1;j<=n;j++)d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
  }
}
```

5.6 2-SAT

设一共有 n 个变量，对于一个变量 i，i 点表示它为 0，$i+n$ 点表示它为 1，$vis[i]$ 表示 i 点选不选。

```
int q[N<<1],t;bool vis[N<<1];
bool dfs(int x){
  if(vis[x>n?x-n:x+n])return 0;
  if(vis[x])return 1;
  vis[q[++t]=x]=1;
  for(int i=g[x];i;i=nxt[i])if(!dfs(v[i]))return 0;
  return 1;
}
bool solve(){
  for(int i=1;i<=n;i++)if(!vis[i]&&!vis[i+n]){
    t=0;
    if(!dfs(i)){
      while(t)vis[q[t--]]=0;
      if(!dfs(i+n))return 0;
    }
  }
  return 1;
}
```

5.7 完美消除序列

一个无向图称为弦图当图中任意长度大于 3 的环都至少有一个弦。

弦图的方法有着很多经典用途：例如用最少的颜色给每个点染色使得相邻的点染的颜色不同，通过完美消除序列从后往前依次给每个点染色，给每个点染上可以染的最小的颜色；最大独立集问题，选择最多的点使得任意两个点不相邻，通过完美消除序列从前往后能选就选。

给定一张 n 个点 m 条边的弦图，求把点最少分成多少组，使得每组点之间没有边。从后往前求完美消除序列。必要时请加上优先队列优化，时间复杂度 $O((n+m)\log n)$。

```
int i,j,k,col[N],ans;bool vis[N];
int main(){
  for(i=n;i;i--){
    for(k=0,j=1;j<=n;j++)if(!vis[j]&&col[j]>=col[k])k=j;
    for(vis[k]=1,j=g[k];j;j=nxt[j])if(!vis[v[j]])if(++col[v[j]]>ans)ans=col[v[j]];
  }
  printf("%d",ans+1);
}
```

5.8 最大团

5.8.1 搜索

```
int n,i,j,k,max[N],g[N][N],f[N][N],ans;
int dfs(int cur,int tot) {
  if(!cur){
    if(tot>ans)return ans=tot,1;
    return 0;
  }
  for(int i=0,j,u,nxt;i<cur;i++){
    if(cur-i+tot<=ans)return 0;
    u=f[tot][i],nxt=0;
    if(max[u]+tot<=ans)return 0;
    for(j=i+1;j<cur;j++)if(g[u][f[tot][j]])f[tot+1][nxt++]=f[tot][j];
    if(dfs(nxt,tot+1))return 1;
  }
  return 0;
}
int main(){
  scanf("%d",&n);
  while(scanf("%d%d",&i,&j)!=EOF)g[i-1][j-1]=g[j-1][i-1]=1;
  for(i=n-1;~i;dfs(k,1),max[i--]=ans)for(k=0,j=i+1;j<n;j++)if(g[i][j])f[1][k++]=j;
  printf("%d",ans);
}
```

5.8.2 随机贪心

```
int T,n,m,i,j,k,g[N][N],a[N],del[N],ans,fin[N];
void solve(){
  for(i=0;i<n;i++)del[i]=0;
  for(k=i=0;i<n;i++)if(!del[i])for(k++,j=i+1;j<n;j++)if(!g[a[i]][a[j]])del[j]=1;
  if(k>ans)for(ans=k,i=j=0;i<n;i++)if(!del[i])fin[j++]=a[i];
}
int main(){
  scanf("%d%d",&n,&m);
  for(i=0;i<n;i++)a[i]=i;
  while(m--)scanf("%d%d",&i,&j),g[i][j]=g[j][i]=1;
  for(T=100;T--;solve())for(i=0;i<n;i++)swap(a[i],a[rand()%n]);
  for(printf("%d\n",ans),i=0;i<ans;i++)printf("%d ",fin[i]+1);
}
```

5.9 最大独立集的随机算法

```
int T,n,i,k,m,x,y,ans,q[N],t,loc[N],del[N],have;
int main(){
  for(T=1000;T;T--){
    for(have=0,t=n,i=1;i<=n;i++)q[i]=loc[i]=i,del[i]=0;
    while(t){
      y=q[x=std::rand()%t+1],loc[q[x]=q[t--]]=x,have++;
      for(p=g[y];p;p=p->nxt)if(!del[p->v])del[p->v]=1,x=loc[p->v],loc[q[x]=q[t--]]=x;
    }
```

```
    if(have>ans)ans=have;
  }
  printf("%d",ans);
}
```

5.10 差分约束系统

a 向 b 连一条权值为 c 的有向边表示 $b-a \le c$，用 SPFA 判断是否存在负环，存在即无解。

5.11 点覆盖、独立集、最大团、路径覆盖

二分图最小路径覆盖 = 最大独立集 = 总节点数 -最大匹配数，最小点覆盖 = 最大匹配数。任意图中，最大独立集 + 最小点覆盖集 =V，最大团 = 补图的最大独立集。

5.12 匈牙利算法

```
bool find(int x){
  for(int i=g[x];i;i=nxt[i])if(!b[v[i]]){
    b[v[i]]=1;
    if(!f[v[i]]||find(f[v[i]]))return f[v[i]]=x,1;
  }
  return 0;
}
int main(){
  for(j=1;j<=m;j++)f[j]=0;
  for(i=1;i<=n;i++){
    for(j=1;j<=m;j++)b[j]=0;
    if(find(i))ans++;
  }
}
```

5.13 Hall 定理

二分图中的两部分顶点组成的集合分别为 X,Y，则有一组无公共点的边，一端恰好为组成 X 的点的充分必要条件是：X 中的任意 k 个点至少与 Y 中的 k 个点相邻。对于区间图只需要考虑极端情况，线段树维护。

5.14 网络流

5.14.1 ISAP 求最大流

```
const int N=410,inf=~0U>>2;
struct edge{int t,f;edge*nxt,*pair;}*g[N],*d[N],pool[M],*cur=pool;
int n,m,i,S,T,h[N],gap[N],maxflow;
void add(int s,int t,int f){
  edge*p=cur++;p->t=t;p->f=f;p->nxt=g[s];g[s]=p;
  p=cur++;p->t=s;p->f=0;p->nxt=g[t];g[t]=p;
  g[s]->pair=g[t];g[t]->pair=g[s];
}
```

```
int sap(int v,int flow){
  if(v==T)return flow;
  int rec=0;
  for(edge*p=d[v];p;p=p->nxt)if(h[v]==h[p->t]+1&&p->f){
    int ret=sap(p->t,min(flow-rec,p->f));
    p->f-=ret;p->pair->f+=ret;d[v]=p;
    if((rec+=ret)==flow)return flow;
  }
  if(!(--gap[h[v]]))h[S]=T;
  gap[++h[v]]++;d[v]=g[v];
  return rec;
}
int main(){
  S=n+1,T=S+1;
  for(cur=pool,i=1;i<=T;i++)g[i]=d[i]=NULL,h[i]=gap[i]=0;
  addedge;
  for(gap[maxflow=0]=T,i=1;i<=T;i++)d[i]=g[i];
  while(h[S]<T)maxflow+=sap(S,inf);
}
```

5.14.2 上下界有源汇网络流

T 向 S 连容量为正无穷的边，将有源汇转化为无源汇。

每条边容量减去下界，设 $in[i]$ 表示流入 i 的下界之和减去流出 i 的下界之和。

新建超级源汇 SS,TT，对于 $in[i]>0$ 的点，SS 向 i 连容量为 $in[i]$ 的边。对于 $in[i]<0$ 的点，i 向 TT 连容量为 $-in[i]$ 的边。

求出以 SS,TT 为源汇的最大流，如果等于 $\sum in[i](in[i]>0)$，则存在可行流。再求出以 S,T 为源汇的最大流即为最大流。

费用流：建完图后等价于求以 SS,TT 为源汇的的费用流。

5.14.3 费用流

最小费用流：若 $d[T]$ 为负则继续增广。

最小费用最大流：若 $d[T]$ 不为 inf 则继续增广。

```
const int inf=~0U>>2,N=210,M=20000;
int n,m,i,tmp,ans;
int u[M],v[M],c[M],co[M],nxt[M],t=1,S,T,l,r,q[M],g[N],f[N],d[N];bool in[N];
void add(int x,int y,int z,int zo){
  u[++t]=x;v[t]=y;c[t]=z;co[t]=zo;nxt[t]=g[x];g[x]=t;
  u[++t]=y;v[t]=x;c[t]=0;co[t]=-zo;nxt[t]=g[y];g[y]=t;
}
bool spfa(){
  int x,i;
  for(i=1;i<=T;i++)d[i]=inf,in[i]=0;
  d[S]=0;in[S]=1;l=r=M>>1;q[l]=S;
  while(l<=r){
    int x=q[l++];
    if(x==T)continue;
    for(i=g[x];i;i=nxt[i])if(c[i]&&co[i]+d[x]<d[v[i]]){
      d[v[i]]=co[i]+d[x];f[v[i]]=i;
      if(!in[v[i]]){
```

```
        in[v[i]]=1;
        if(d[v[i]]<d[q[l]])q[--l]=v[i];else q[++r]=v[i];
      }
    }
    in[x]=0;
  }
  return d[T]<inf;
}
int main(){
  S=0,T=n+1;
  while(spfa()){
    for(tmp=inf,i=T;i!=S;i=u[f[i]])if(tmp>c[f[i]])tmp=c[f[i]];
    for(ans+=d[i=T]*tmp;i!=S;i=u[f[i]])c[f[i]]-=tmp,c[f[i]^1]+=tmp;
  }
  printf("%d",ans);
}
```

5.14.4 混合图欧拉回路判定

首先给无向边随便定一个方向，设 $deg[x]$ 为 x 连出去的边数 − 连入 x 的边数。

若存在 $deg[x]$ 为奇数，或者图不连通，则无解。否则建立源点 S，汇点 T。

对于一个点 x，若 $deg[x] > 0$，则 S 向 x 连边，容量 $\frac{deg[x]}{2}$；若 $deg[x] < 0$，则 x 向 T 连边，容量 $-\frac{deg[x]}{2}$。

对于一条定了向的无向边 x->y，x 向 y 连边，容量 1，求出最大流，若与 S 和 T 连的每条边都满流，则有解。

5.14.5 线性规划转费用流

首先添加松弛变量，将不等号都变为等号。分别用下一个式子减去上一个式子，如果每个变量只出现了两次且符号一正一负，那么可以转化为费用流。对于每个式子建立一个点，那么每个变量对应一条边，从一个点流出，向另一个点流入。这样，对于等式右边的常数 C，如果是正的，对应从源点向该点连一条流量 C，费用 0 的边；如果是负的对应从该点向汇点连一条流量 $-C$，费用 0 的边。对于每个变量，从它系数为正的式子向系数为负的式子连一条容量为 inf，费用为它在目标函数里系数的边。这样网络流模型就构造完毕了。

5.15 最小树形图

```
const int N=10050,M=50050,inf=0x7fffffff;
struct DMST{
  int n,size,pre[N],id[N],vis[N],in[N];
  struct EDGE{
    int u,v,cost;
    EDGE(){}
    EDGE(int a,int b,int c):u(a),v(b),cost(c){}
  }E[M];
  void init(int _n){n=_n,size=0;}
  void add(int u,int v,int w){E[size++]=EDGE(u,v,w);}
  int dmst(int root){
    int u,v,cnt,ret=0;
```

```
    while(1){
      for(int i=0;i<n;i++)in[i]=inf;
      for(int i=0;i<size;i++){
        u=E[i].u,v=E[i].v;
        if(E[i].cost<in[v]&&u!=v){
          pre[v]=u,in[v]=E[i].cost;
          if(u==root)ROOT=i;
        }
      }
      for(int i=0;i<n;i++)if(i!=root&&in[i]==inf)return -1;
      cnt=in[root]=0;
      for(i=0;i<n;i++)id[i]=vis[i]=-1;
      for(int i=0;i<n;i++){
        ret+=in[i],v=i;
        while(vis[v]!=i&&id[v]==-1&&v!=root)vis[v]=i,v=pre[v];
        if(v!=root&&id[v]==-1){
          for(u=pre[v];u!=v;u=pre[u])id[u]=cnt;
          id[v]=cnt++;
        }
      }
      if(!cnt)break;
      for(int i=0;i<n;i++)if(id[i]==-1)id[i]=cnt++;
      for(int i=0;v=E[i].v,i<size;i++){
        E[i].u=id[E[i].u],E[i].v=id[E[i].v];
        if(E[i].u!=E[i].v)E[i].cost-=in[v];
      }
      n=cnt,root=id[root];
    }
    return ret;
  }
};
void variable(int &cost,int &root){//Variable Root
  for(int i=0;i<n;i++)G.add(st,i,tot);//st=n,tot=sum of Edge Wight+1
  int ans=G.dmst(st);
  if(ans==-1||ans-tot>tot)return;//No solution
  cost=ans-tot,root=ROOT-m;
}
```

5.16 构造双连通图

一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为 1。

统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 $leaf$。则至少在树上添加 $\frac{leaf+1}{2}$ 条边，就能使树达到边双连通。具体方法为，首先把两个最近公共祖先最远的两个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是 $\frac{leaf+1}{2}$ 次，把所有点收缩到了一起。

5.17 一般图最大匹配

用带花树求编号从 0 开始的 n 个点的图的最大匹配，时间复杂度 $O(n^3)$。

$mate[]$ 为配偶结点的编号，没有匹配上的点为 -1。

传入结点个数 n 及各结点的出边表 $G[]$, 返回匹配点对的数量 ret。

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
#include<queue>
using namespace std;
const int N=510;
int n,m,x,y;vector<int>g[N];
namespace Blossom{
int mate[N],n,ret,nxt[N],f[N],mark[N],vis[N],t;queue<int>Q;
int F(int x){return x==f[x]?x:f[x]=F(f[x]);}
void merge(int a,int b){f[F(a)]=F(b);}
int lca(int x,int y){
  for(t++;;swap(x,y))if(~x){
    if(vis[x=F(x)]==t)return x;vis[x]=t;
    x=mate[x]!=-1?nxt[mate[x]]:-1;
  }
}
void group(int a,int p){
  for(int b,c;a!=p;merge(a,b),merge(b,c),a=c){
    b=mate[a],c=nxt[b];
    if(F(c)!=p)nxt[c]=b;
    if(mark[b]==2)mark[b]=1,Q.push(b);
    if(mark[c]==2)mark[c]=1,Q.push(c);
  }
}
void aug(int s,const vector<int>G[]){
  for(int i=0;i<n;++i)nxt[i]=vis[i]=-1,f[i]=i,mark[i]=0;
  while(!Q.empty())Q.pop();Q.push(s);mark[s]=1;
  while(mate[s]==-1&&!Q.empty()){
    int x=Q.front();Q.pop();
    for(int i=0,y;i<(int)G[x].size();++i){
      if((y=G[x][i])!=mate[x]&&F(x)!=F(y)&&mark[y]!=2){
        if(mark[y]==1){
          int p=lca(x,y);
          if(F(x)!=p)nxt[x]=y;
          if(F(y)!=p)nxt[y]=x;
          group(x,p),group(y,p);
        }else if(mate[y]==-1){
          nxt[y]=x;
          for(int j=y,k,l;~j;j=l)k=nxt[j],l=mate[k],mate[j]=k,mate[k]=j;
          break;
        }else nxt[y]=x,Q.push(mate[y]),mark[mate[y]]=1,mark[y]=2;
      }
    }
  }
}
int solve(int _n,const vector<int>G[]){
  n=_n;memset(mate,-1,sizeof mate);
  for(int i=t=0;i<n;++i)if(mate[i]==-1)aug(i,G);
```

```
  for(int i=ret=0;i<n;++i)ret+=mate[i]>i;
  printf(”%d\n”,ret);
  for(int i=0;i<n;i++)printf(”%d ”,mate[i]+1);
  return ret;
}
}
int main(){
  scanf(”%d%d”,&n,&m);
  while(m——)scanf(”%d%d”,&x,&y),x——,y——,g[x].push_back(y),g[y].push_back(x);
  Blossom::solve(n,g);
}
```

6 博弈论

6.1 树上删边游戏

给定一棵 n 个点的有根树，每次可以删掉一个子树，则叶子节点的 SG 值为 0，非叶子节点的 SG 值为其所有孩子节点 (SG 值 +1) 的异或和。

7 数学

7.1 Bell 数

B_n 表示把 n 个带标号的物品划分为若干不相交集合的方案数。

有递推式 $B_{n+1} = \sum_{k=0}^{n} C(n,k)B_k$。

下面为 $O(P^2 \log P)$ 计算 B_n 对 $999999598 = 2 \times 13 \times 5281 \times 7283$ 取模的代码。

```
#include<cstdio>
typedef long long ll;
const int N=7284,P=999999598;
ll n;int a[4]={2,13,5281,7283},f[N],s[2][N],i,j,x;
int cal(int x,ll n){
  int i,j,k,m=0,b[N],c[N],d[70];
  for(i=0;i<=x;i++)b[i]=f[i]%x;
  while(n)d[m++]=n%x,n/=x;
  for(i=1;i<m;i++)for(j=1;j<=d[i];j++){
    for(k=0;k<x;k++)c[k]=(b[k]*i+b[k+1])%x;
    c[x]=(c[0]+c[1])%x;
    for(k=0;k<=x;k++)b[k]=c[k];
  }
  return c[d[0]];
}
ll pow(ll a,ll b,ll p){ll t=1;for(a%=p;b;b>>=1LL,a=a*a%p)if(b&1LL)t=t*a%p;return t;}
ll bell(ll n){
  if(n<N)return f[n];
  ll t=0;
  for(int i=0;i<4;i++)t=(t+(P/a[i])*pow(P/a[i],a[i]-2,a[i])%P*cal(a[i],n)%P)%P;
  return t;
}
int main(){
  f[0]=f[1]=s[0][0]=1,s[0][1]=2;
  for(i=2,x=1;i<N;i++,x^=1)for(f[i]=s[x][0]=s[x^1][i-1],j=1;j<=i;j++)
    s[x][j]=(s[x^1][j-1]+s[x][j-1])%P;
  scanf("%lld",&n),printf("%lld",bell(n));
}
```

7.2 扩展欧几里得算法解同余方程

$ans[]$ 保存的为循环节内的所有解。

```
int exgcd(int a,int b,int&x,int&y){
  if(!b)return x=1,y=0,a;
  int d=exgcd(b,a%b,x,y),t=x;
  return x=y,y=t-a/b*y,d;
}
void cal(ll a,ll b,ll n){//ax=b(mod n)
  ll x,y,d=exgcd(a,n,x,y);
  if(b%d)return;
  x=(x%n+n)%n;
  ans[cnt=1]=x*(b/d)%(n/d);
  for(ll i=1;i<d;i++)ans[++cnt]=(ans[1]+i*n/d)%n;
}
```

7.3 同余方程组

```
int n,flag,k,m,a,r,d,x,y;
int main(){
  scanf("%d",&n);
  flag=k=1,m=0;
  while(n--){
    scanf("%d%d",&a,&r);//ans%a=r
    if(flag){
      d=exgcd(k,a,x,y);
      if((r-m)%d){flag=0;continue;}
      x=(x*(r-m)/d+a/d)%(a/d),y=k/d*a,m=((x*k+m)%y)%y;
      if(m<0)m+=y;
      k=y;
    }
  }
  printf("%d",flag?m:-1);//若flag=1，说明有解，解为ki+m，i为任意整数
}
```

7.4 线性基

7.4.1 异或线性基

若要查询第 k 小子集异或和，则把 k 写成二进制，对于是 1 的第 i 位，把从低位到高位第 i 个不为 0 的数异或进答案。

若要判断是否有非空子集的异或和为 0，如果不存在自由基，那么说明只有空集的异或值为 0，需要高斯消元来判断。

```
struct Base{
  int a[31];
  Base(){for(int i=0;i<31;i++)a[i]=0;}
  void ins(int x){for(int i=30;~i;i--)if(x>>i&1){if(a[i])x^=a[i];else{a[i]=x;break;}}}
  void ask(){//查询最大子集异或和
    int t=0;
    for(int i=30;~i;i--)up(t,t^a[i]);
    printf("%d\n",t);
  }
};
```

7.4.2 实数线性基

ins 返回要插入的数是否可以被之前的数线性表示出来，返回 1 表示不能，0 表示可以。

```
struct Base{
  double a[N][N];bool v[N];
  Base(){
    for(int i=0;i<N;i++)for(int j=0;j<N;j++)a[i][j]=0;
    for(int i=0;i<N;i++)v[i]=0;
  }
  bool ins(double*x){
    for(int i=0;i<m;i++)if(fabs(x[i])>1e-5){
      if(v[i]){
```

```
      double t=x[i]/a[i][i];
      for(int j=0;j<m;j++)x[j]-=t*a[i][j];
    }else{
      v[i]=1;
      for(int j=0;j<m;j++)a[i][j]=x[j];
      return 1;
    }
  }
  return 0;
 }
};
```

7.5 原根、指标、离散对数

设 P 为质数，G 为 P 的原根，则 $x^y \equiv b \pmod P$ 等价于 $y\ ind\ x \equiv b \pmod{P-1}$。其中 $G^{ind\ x} \equiv x \pmod P$。

7.5.1 求原根

```
int q[10000];
int pow(ll a,int b,int P){ll t=1;for(;b;b>>=1,a=(ll)a*a%P)if(b&1)t=t*a%P;return t;}
int getG(int n){
  int i,j,t=0;
  for(i=2;(ll)i*i<n-1;i++)if((n-1)%i==0)q[t++]=i,q[t++]=(n-1)/i;
  for(i=2;;i++){
    for(j=0;j<t;j++)if(pow(i,q[j],n)==1)break;
    if(j==t)return i;
  }
}
```

7.5.2 扩展 Baby Step Giant Step

求满足 $a^x \bmod m = r$ 的最小整数 x。

```
#include<cstdio>
#include<cmath>
#include<map>
#include<algorithm>
#include<tr1/unordered_map>
using namespace std::tr1;
using namespace std;
typedef long long ll;
typedef pair<int,int>P;
int phi(int n){
  int t=1,i;
  for(i=2;i*i<=n;i++)if(n%i==0)for(n/=i,t*=i-1;n%i==0;n/=i,t*=i);
  if(n>1)t*=n-1;
  return t;
}
int pow(ll a,int b,int m){ll t=1;for(;b;b>>=1,a=a*a%m)if(b&1)t=t*a%m;return t;}
int bsgs(int a,int r,int m){
  if(r>=m)return -1;
```

```
  int i,g,x,c=0,at=int(2+sqrt(m));
  for(i=0,x=1%m;i<50;i++,x=ll(x)*a%m)if(x==r)return i;
  for(g=x=1;__gcd(int(ll(x)*a%m),m)!=g;c++)g=__gcd(x=ll(x)*a%m,m);
  if(r%g)return -1;
  if(x==r)return c;
  unordered_map<int,int>u;
  g=phi(m/g),u[x]=0;g=pow(a,g-at%g,m);
  for(i=1;i<at;i++){
    u.insert(P(x=ll(x)*a%m,i));
    if(x==r)return c+i;
  }
  for(i=1;i<at;i++){
    unordered_map<int,int>::iterator t=u.find(r=ll(r)*g%m);
    if(t!=u.end())return c+i*at+t->second;
  }
  return -1;
}
```

7.6 Catalan 数

$h_1=1$，$h_n=\frac{h_{n-1}(4n-2)}{n+1}=\frac{C(2n,n)}{n+1}=C(2n,n)-C(2n,n-1)$。

在一个格点阵列中，从 $(0,0)$ 点走到 (n,m) 点且不经过对角线 $x=y$ 的方法数 $(x>y)$：$C(n+m-1,m)-C(n+m-1,m-1)$。

在一个格点阵列中，从 $(0,0)$ 点走到 (n,m) 点且不穿过对角线 $x=y$ 的方法数 $(x\geq y)$：$C(n+m,m)-C(n+m,m-1)$。

7.7 扩展 Cayley 公式

对于 n 个点，m 个连通块的图，假设每个连通块有 $a[i]$ 个点，那么用 $s-1$ 条边把它连通的方案数为 $n^{s-2}a[1]a[2]...a[m]$。

7.8 Jacobi's Four Square Theorem

设 $a^2+b^2+c^2+d^2=n$ 的自然数解个数为 $r4(n)$，$d(n)$ 为 n 的约数和，由 Jacobi's Four Square Theorem 可知，若 n 是奇数，则 $r4(n)=8d(n)$，否则 $r4(n)=24d(k)$，k 是 n 去除所有 2 后的结果。

7.9 复数

复数相乘的几何意义为长度相乘，极角相加。

```
struct comp{
double r,i;comp(double _r=0,double _i=0){r=_r;i=_i;}
comp operator+(const comp x){return comp(r+x.r,i+x.i);}
comp operator-(const comp x){return comp(r-x.r,i-x.i);}
comp operator*(const comp x){return comp(r*x.r-i*x.i,r*x.i+i*x.r);}
comp operator/(const comp x){
  double t=x.r*x.r+x.i*x.i;
  return comp((r*x.r+i*x.i)/t,(i*x.r-r*x.i)/t);
}};
```

7.10 高斯消元

n 个未知量，n 个方程，a 为增广矩阵。

```
for(i=1;i<=n;i++){
  for(k=i,j=i+1;j<=n;j++)if(fabs(a[j][i])>fabs(a[k][i]))k=j;
  if(k!=i)for(j=i;j<=n+1;j++)t=a[i][j],a[i][j]=a[k][j],a[k][j]=t;
  for(j=i+1;j<=n;j++)for(t=a[j][i]/a[i][i],k=i;k<=n+1;k++)a[j][k]-=a[i][k]*t;
}
for(ans[n]=a[n][n+1]/a[n][n],i=n-1;i;i--){
  for(ans[i]=a[i][n+1],j=n;j>i;j--)ans[i]-=ans[j]*a[i][j];
  ans[i]/=a[i][i];
}
```

7.10.1 行列式

求矩阵 a 的 n 阶行列式对任意数 P 取模的结果。

```
ll det(int n){
  ll ans=1;bool flag=1;
  for(i=1;i<=n;i++)for(j=1;j<=n;j++)a[i][j]=(a[i][j]+P)%P;
  for(i=1;i<=n;i++){
    for(j=i+1;j<=n;j++)while(a[j][i]){
      ll t=a[i][i]/a[j][i];
      for(k=i;k<=n;k++)a[i][k]=(a[i][k]+P-t*a[j][k]%P)%P;
      for(k=i;k<=n;k++)swap(a[i][k],a[j][k]);
      flag^=1;
    }
    ans=ans*a[i][i]%P;
    if(!ans)return 0;
  }
  if(!flag)ans=(P-ans);
  return ans;
}
```

7.10.2 Matrix-Tree 定理

对于一张图，建立矩阵 C，$C[i][i]=i$ 的度数，若 i,j 之间有边，那么 $C[i][j]=-1$，否则为 0。这张图的生成树个数等于矩阵 C 的 $n-1$ 阶行列式的值。

7.11 康托展开

输入 n，查询某个排列的排名以及字典序第 m 小的排列。

```
#include<cstdio>
int n,q,i,j,t,a[22];long long f[22],m,ans;char op[5];
int main(){
  scanf("%d%d",&n,&q);
  for(f[1]=1,i=2;i<n;i++)f[i]=f[i-1]*i;
  while(q--){
    scanf("%s",op);
    if(op[0]=='P'){
      scanf("%lld",&m);
```

```
    for(i=1;i<=n;i++)a[i]=i;
    for(m--,j=1;j<n;j++){
      printf("%d ",a[m/f[n-j]+1]);
      for(i=m/f[n-j]+1;i<=n-j;i++)a[i]=a[i+1];
      m%=f[n-j];
    }
    printf("%d\n",a[1]);
  }else{
    for(i=1;i<=n;i++)scanf("%d",&a[i]);
    for(ans=0,i=1;i<n;i++){
      for(t=0,j=n;j>i;j--)if(a[j]<a[i])t++;
      ans=(ans+t)*(n-i);
    }
    printf("%lld\n",ans+1);
  }
 }
}
```

7.12 自适应 Simpson

给定一个函数 $f(x)$，求 $[a,b]$ 区间内 $f(x)$ 到 x 轴所形成区域的面积。

根据辛普森公式，有 S 近似等于 $\frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$。

```
double simpson(double l,double r){return (f(l)+f(r)+4*f((l+r)/2.0))*(r-l)/6.0;}
double rsimpson(double l,double r){
  double mid=(l+r)/2.0;
  if(fabs(simpson(l,r)-simpson(l,mid)-simpson(mid,r))<eps)
    return simpson(l,mid)+simpson(mid,r);
  return rsimpson(l,mid)+rsimpson(mid,r);
}
```

7.13 线性规划

n 个约束条件，m 个未知数，求 $\sum_{i=1}^{m} a[0][i]x[i]$ 的最大值。

约束条件：$\sum_{j=1}^{m}(-a[i][j])\times x[j]\le a[i][0]$。

若要求最小值，则需进行对偶，即把目标函数的系数与约数条件右边的数交换，然后把矩阵转置。

```
#define rep(i,l,n) for(int i=l;i<=n;i++)
const int N=1810,M=610,inf=~0U>>2;
int n,m,a[N][M],nxt[M];
void cal(int l,int e){
  a[l][e]=-1;t=M-1;
  rep(i,0,m)if(a[l][i])nxt[t]=i,t=i;nxt[t]=-1;
  rep(i,0,n)if(i!=l&&(t=a[i][e])){
    a[i][e]=0;
    for(int j=nxt[M-1];~j;j=nxt[j])a[i][j]+=a[l][j]*t;
  }
}
int work(){
  for(;;){int min=inf,l=0,e=0;
    rep(i,1,m)if(a[0][i]>0){e=i;break;}
```

```
    if(!e)return a[0][0];
    rep(i,1,n)if(a[i][e]<0&&a[i][0]<min)min=a[i][0],l=i;
    cal(l,e);
  }
}
```

7.14 调和级数

$\sum_{i=1}^{n}\frac{1}{i}$ 在 n 较大时约等于 $\ln n+r$，r 为欧拉常数，约等于 0.5772156649015328。

7.15 曼哈顿距离的变换

$|x_1-x_2|+|y_1-y_2|=\max(|(x_1+y_1)-(x_2+y_2)|,|(x_1-y_1)-(x_2-y_2)|)$

7.16 拉格朗日乘数法

偏导数：

对于一个多元函数，关于某个元 x 的偏导，就是将其他的量看作常数，然后这个多元函数就很显然地变成了一元函数，之后我们对这个一元函数求导，导数就是原多元函数关于 x 的偏导了。

拉格朗日乘数法：

设 $\bigtriangledown f$ 表示 f 的偏导，对于多元函数 f 和约束条件 g，多元函数 f 取到极值，当且仅当存在负数 λ，使得 f 关于每个元的偏导 $\bigtriangledown f$，以及对应的 g 都满足，$\bigtriangledown f=\lambda\times\bigtriangledown g$。

7.17 线性递推逆元

```
for(r[0]=r[1]=1,i=2;i<P;i++){
  r[i]=-r[P%i]*(P/i)%P;
  while(r[i]<0)r[i]+=P;
}
```

7.18 组合数取模

7.18.1 Lucas 定理

```
#include<cstdio>
#define P 10007
int T,n,m,i,f[P],r[P];
int C(int n,int m){return n<m?0:f[n]*r[n-m]%P*r[m]%P;}
int lucas(int n,int m){
  if(n<m)return 0;
  if(!m||n==m)return 1;
  return C(n%P,m%P)*lucas(n/P,m/P)%P;
}
int main(){
  for(r[0]=r[1]=f[0]=f[1]=1,i=2;i<P;i++){
    f[i]=f[i-1]*i%P,r[i]=-r[P%i]*(P/i)%P;
    while(r[i]<0)r[i]+=P;
```

```
  }
  for(i=2;i<P;i++)r[i]=r[i]*r[i-1]%P;
  for(scanf("%d",&T);T--;printf("%d\n",lucas(n,m)))scanf("%d%d",&n,&m);
}
```

7.18.2 P 是质数的幂

B 表示质数，P 表示模数，$cal(n)$ 将返回 $n!$，以 $a \times B^b$ 形式表示。

```
ll n,x,y,P,B,s[2000000];
ll exgcd(ll a,ll b){
  if(!b)return x=1,y=0,a;
  ll d=exgcd(b,a%b),t=x;
  return x=y,y=t-a/b*y,d;
}
ll rev(ll a,ll P){exgcd(a,P);while(x<0)x+=P;return x%P;}
ll pow(ll a,ll b,ll P){ll t=1;for(;b;b>>=1LL,a=a*a%P)if(b&1LL)t=t*a%P;return t;}
struct Num{
  ll a,b;
  Num(){a=1,b=0;}
  Num(ll _a,ll _b){a=_a,b=_b;}
  Num operator*(Num x){return Num(a*x.a%P,b+x.b);}
  Num operator/(Num x){return Num(a*rev(x.a,P)%P,b-x.b);}
}now[2];
Num cal(ll n){return n?Num(s[n%P]*pow(s[P],n/P,P)%P,n/B)*cal(n/B):Num(1,0);}
void pre(){for(i=s[0]=1;i<P;i++)if(i%B)s[i]=s[i-1]*i%P;else s[i]=s[i-1];s[P]=s[P-1];}
int main(){
  B=2,P=512,pre();
  cal(n);
}
```

7.19 超立方体相关

n 维超立方体有 $2^{n-i} \times C(n,i)$ 个 i 维元素。

7.20 平面图欧拉公式

对于连通的平面图，有区域数 $F =$ 点数 $E-$ 边数 $V + 1$。

7.21 线性筛

对于线性筛求积性函数，只需考虑质数的情况，质数的幂的情况即可。

```
for(mu[1]=phi[1]=1,i=2;i<N;i++){
  if(!v[i])p[tot++]=i,mu[i]=-1,phi[i]=i-1;
  for(j=0;i*p[j]<N&&j<tot;j++){
    v[i*p[j]]=1;
    if(i%p[j]){
      mu[i*p[j]]=-mu[i];
      phi[i*p[j]]=phi[i]*(p[j]-1);
    }else{
      mu[i*p[j]]=0;
```

```
      phi[i*p[j]]=phi[i]*p[j];
      break;
    }
  }
}
```

7.22 数论函数变换

常见积性函数：

$id(i) = i$

$e(i) = [i = 1]$

$d(i) = i$ 的约数个数

$\sigma(i) = i$ 的约数之和

一些性质：

$n = \sum_{d|n} \varphi(d)$

$e(n) = \sum_{d|n} \mu(d)$

$\sum_{i=1}^{n} \sum_{j=1}^{m} i \times j[\gcd(i,j) = d] = \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} id \times jd[\gcd(i,j) = 1]$

$\mu \times id = \varphi$

莫比乌斯反演：

$f(n) = \sum_{d|n} g(d)$

$g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$

7.22.1 疯狂的前缀和

对于快速计算 φ 的前缀和：

$S_n = \frac{n(n+1)}{2} - \sum_{d=1}^{n} S_{\frac{n}{d}}$

对于快速计算 μ 的前缀和：

$S_n = 1 - \sum_{d=1}^{n} S_{\frac{n}{d}}$

n 小于 $maxn^{\frac{2}{3}}$ 时线性筛预处理，否则记忆化搜索，单次计算时间复杂度 $O(n^{\frac{2}{3}})$。

```
#include<cstdio>
#include<algorithm>
#include<map>
#define M 1670000
using namespace std;
typedef unsigned long long ll;
struct P{
  ll x;int y;
  P(){x=y=0;}
  P(ll _x,int _y){x=_x,y=_y;}
  void operator+=(P b){x+=b.x,y+=b.y;}
  void operator-=(P b){x-=b.x,y-=b.y;}
  P operator*(int b){return P(x*b,y*b);}
  void write(){printf("%llu %d\n",x,y);}
}pre[M];
int n,i,j,p[M],tot,N,ask[10],Q;bool v[M];map<int,P>T;
P sum(int n){
```

```
  if(n<N)return pre[n];
  if(T.find(n)!=T.end())return T[n];
  P t=P(((ll)n+1)*n/2,1);
  for(int i=2,j=0;i<=n&&j<n;i=j+1)t-=sum(n/i)*((j=n/(n/i))-i+1);
  return T[n]=t;
}
int main(){
  for(scanf("%d",&Q);i<Q;n=max(n,ask[i++]))scanf("%d",&ask[i]);
  while((ll)N*N*N<n)N++;N*=N;
  for(pre[1]=P(1,1),i=2;i<N;i++){
    if(!v[i])p[tot++]=i,pre[i]=P(i-1,-1);
    for(j=0;i*p[j]<N&&j<tot;j++){
      v[i*p[j]]=1;
      if(i%p[j])pre[i*p[j]]=P(pre[i].x*(p[j]-1),-pre[i].y);
      else{pre[i*p[j]]=P(pre[i].x*p[j],0);break;}
    }
  }
  for(i=2;i<=N;i++)pre[i]+=pre[i-1];
  for(i=0;i<Q;i++)sum(ask[i]).write();
}
```

7.23 快速傅里叶变换

下列模板中 n 必须为 2 的幂。

7.23.1 FFT

```
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
struct comp{
  double r,i;comp(double _r=0,double _i=0){r=_r;i=_i;}
  comp operator+(const comp x){return comp(r+x.r,i+x.i);}
  comp operator-(const comp x){return comp(r-x.r,i-x.i);}
  comp operator*(const comp x){return comp(r*x.r-i*x.i,r*x.i+i*x.r);}
};
const double pi=acos(-1.0);
void FFT(comp a[],int n,int t){
  for(int i=1,j=0;i<n-1;i++){
    for(int s=n;j^=s>>=1,~j&s;);
    if(i<j)swap(a[i],a[j]);
  }
  for(int d=0;(1<<d)<n;d++){
    int m=1<<d,m2=m<<1;
    double o=pi/m*t;comp _w(cos(o),sin(o));
    for(int i=0;i<n;i+=m2){
      comp w(1,0);
      for(int j=0;j<m;j++){
        comp &A=a[i+j+m],&B=a[i+j],t=w*A;
        A=B-t;B=B+t;w=w*_w;
      }
    }
  }
```

```
  if(t==-1)for(int i=0;i<n;i++)a[i].r/=n;
}
```

7.23.2 NTT

998244353 原根为 3，1004535809 原根为 3，786433 原根为 10，880803841 原根为 26。

```
#include<cstdio>
typedef long long ll;
const int N=262144,K=17;
int n,m,i,k;
int a[N+10],b[N+10],tmp[N+10],tmp2[N+10];
int P=998244353,G=3,g[K+1],ng[K+10],inv[N+10],inv2;
int pow(int a,int b){int t=1;for(;b;b>>=1,a=(ll)a*a%P)if(b&1)t=(ll)t*a%P;return t;}
void NTT(int*a,int n,int t){
  for(int i=1,j=0;i<n-1;i++){
    for(int s=n;j^=s>>=1,~j&s;);
    if(i<j){int k=a[i];a[i]=a[j];a[j]=k;}
  }
  for(int d=0;(1<<d)<n;d++){
    int m=1<<d,m2=m<<1,_w=t==1?g[d]:ng[d];
    for(int i=0;i<n;i+=m2)for(int w=1,j=0;j<m;j++){
      int&A=a[i+j+m],&B=a[i+j],t=(ll)w*A%P;
      A=B-t;if(A<0)A+=P;
      B=B+t;if(B>=P)B-=P;
      w=(ll)w*_w%P;
    }
  }
  if(t==-1)for(int i=0,j=inv[n];i<n;i++)a[i]=(ll)a[i]*j%P;
}
//给定a,求a的逆元b
void getinv(int*a,int*b,int n){
  if(n==1){b[0]=pow(a[0],P-2);return;}
  getinv(a,b,n>>1);
  int k=n<<1,i;
  for(i=0;i<n;i++)tmp[i]=a[i];
  for(i=n;i<k;i++)tmp[i]=b[i]=0;
  NTT(tmp,k,1),NTT(b,k,1);
  for(i=0;i<k;i++){
    b[i]=(ll)b[i]*(2-(ll)tmp[i]*b[i]%P)%P;
    if(b[i]<0)b[i]+=P;
  }
  NTT(b,k,-1);
  for(i=n;i<k;i++)b[i]=0;
}
//给定a,求a的对数函数，且a[0]=1
void getln(int*a,int*b,int n){
  getinv(a,tmp2,n);
  int k=n<<1,i;
  for(i=0;i<n-1;i++)b[i]=(ll)a[i+1]*(i+1)%P;
  for(i=n-1;i<k;i++)b[i]=0;
  NTT(b,k,1),NTT(tmp2,k,1);
  for(i=0;i<k;i++)b[i]=(ll)b[i]*tmp2[i]%P;
  NTT(b,k,-1);
  for(i=n-1;i;i--)b[i]=(ll)b[i-1]*inv[i]%P;b[0]=0;
```

```
}
//给定a,求a的指数函数，且a[0]=0
void getexp(int*a,int*b,int n){
  if(n==1){b[0]=1;return;}
  getexp(a,b,n>>1);
  getln(b,tmp,n);
  int k=n<<1,i;
  for(i=0;i<n;i++){tmp[i]=a[i]-tmp[i];if(tmp[i]<0)tmp[i]+=P;}
  if((++tmp[0])==P)tmp[0]=0;
  for(i=n;i<k;i++)tmp[i]=b[i]=0;
  NTT(tmp,k,1),NTT(b,k,1);
  for(i=0;i<k;i++)b[i]=(ll)b[i]*tmp[i]%P;
  NTT(b,k,-1);
  for(i=n;i<k;i++)b[i]=0;
}
//给定a,求a的平方根，b且a[0]=1
void getroot(int*a,int*b,int n){
  if(n==1){b[0]=1;return;}
  getroot(a,b,n>>1);
  getinv(b,tmp2,n);
  int k=n<<1,i;
  for(i=0;i<n;i++)tmp[i]=a[i];
  for(i=n;i<k;i++)tmp[i]=b[i]=0;
  NTT(tmp,k,1),NTT(b,k,1),NTT(tmp2,k,1);
  for(i=0;i<k;i++)b[i]=((ll)b[i]*b[i]+tmp[i])%P*inv2%P*tmp2[i]%P;
  NTT(b,k,-1);
  for(i=n;i<k;i++)b[i]=0;
}
int main(){
  for(g[K]=pow(G,(P-1)/N),ng[K]=pow(g[K],P-2),i=K-1;~i;i--)
    g[i]=(ll)g[i+1]*g[i+1]%P,ng[i]=(ll)ng[i+1]*ng[i+1]%P;
  for(inv[1]=1,i=2;i<=N;i++)inv[i]=(ll)(P-inv[P%i])*(P/i)%P;inv2=inv[2];
  scanf("%d%d",&n,&m);
  for(k=1;k<=n;k<<=1);
  for(i=0;i<=n;i++)scanf("%d",&a[i]);
  getln(a,b,k);
  for(i=0;i<k;i++)b[i]=(ll)b[i]*m%P;
  getexp(b,a,k);
  for(i=0;i<k;i++)printf("%d ",a[i]);
}
```

7.23.3 多项式求幂

找到最低的不为 0 的那一项，把整个多项式除以它，然后 ln+exp 在 $O(n\log n)$ 时间内求出幂，再乘回去即可。

7.23.4 拉格朗日反演

若 F 与 G 互为复合逆，即互为反函数，根据拉格朗日反演可得，
$[x^n]F(x)=[x^{n-1}]\frac{(\frac{x}{G(x)})^n}{n}$，其中 $[x^n]$ 表示第 n 项的系数。

7.24 蔡勒公式

$w = (\lfloor \frac{c}{4} \rfloor - 2c + y + \lfloor \frac{y}{4} \rfloor + \lfloor \frac{13(m+1)}{5} \rfloor + d - 1) \bmod 7$

w：0 星期日，1 星期一，2 星期二，3 星期三，4 星期四，5 星期五，6 星期六。

c：世纪减 1（年份前两位数）。

y：年（后两位数）。

m：月（$3 \le m \le 14$，即在蔡勒公式中，1、2 月要看作上一年的 13、14 月来计算）。

d：日。

7.25 皮克定理

给定顶点坐标均是整点（或正方形格点）的简单多边形，皮克定理说明了其面积 S 和内部格点数目 n、边上格点数目 s 的关系：$S = n + \frac{s}{2} + 1$。

7.26 组合数 lcm

$(n+1)lcm(C(n,0), C(n,1), ..., C(n,k)) = lcm(n+1, n, n-1, n-2, ..., n-k+1)$

7.27 区间 lcm 的维护

对于一个数，将其分解质因数，若有因子 p^k，那么拆分出 k 个数 $p, p^2, ..., p^k$，权值都为 p，那么查询区间 $[l, r]$ 内所有数的 lcm 的答案 = 所有在该区间中出现过的数的权值之积，可持久化线段树维护即可。

7.28 中国剩余定理

n 个同余方程，第 i 个为 $x \equiv b[i] \pmod{a[i]}$，且 $a[i]$ 两两互质，那么可以通过中国剩余定理合并。

```
ll CRT(ll*a,ll*b,int n){
  ll ans,P=1;
  for(int i=0;i<n;i++)P*=a[i];
  for(int i=0;i<n;i++)ans=(ans+(P/a[i])*pow(P/a[i],a[i]-2,a[i])%P*b[i]%P)%P;
  while(ans<0)ans+=P;
  return ans;
}
```

7.29 欧拉函数

```
int phi(int n){
  int t=1,i;
  if(!(n&1))for(n>>=1;!(n&1);n>>=1,t<<=1);
  for(i=3;i*i<=n;i+=2)if(n%i==0)for(n/=i,t*=i-1;n%i==0;n/=i,t*=i);
  if(n>1)t*=n-1;
  return t;
}
```

7.30 快速沃尔什变换

```
void FWT(int*a,int n){
  for(int d=1;d<n;d<<=1)for(int m=d<<1,i=0;i<n;i+=m)for(int j=0;j<d;j++){
    int x=a[i+j],y=a[i+j+d];
    //xor:a[i+j]=x+y,a[i+j+d]=x-y;
    //and:a[i+j]=x+y;
    //or:a[i+j+d]=x+y;
  }
}
void UFWT(int*a,int n){
  for(int d=1;d<n;d<<=1)for(int m=d<<1,i=0;i<n;i+=m)for(int j=0;j<d;j++){
    int x=a[i+j],y=a[i+j+d];
    //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
    //and:a[i+j]=x-y;
    //or:a[i+j+d]=y-x;
  }
}
```

7.31 幂和

$$\begin{aligned}
\sum_{i=1}^{n} i^1 &= \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n \\
\sum_{i=1}^{n} i^2 &= \frac{n(n+1)(2n+1)}{6} = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n \\
\sum_{i=1}^{n} i^3 &= \left[\frac{n(n+1)}{2}\right]^2 = \frac{1}{4}n^4 + \frac{1}{2}n^3 + \frac{1}{4}n^2 \\
\sum_{i=1}^{n} i^4 &= \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}n \\
\sum_{i=1}^{n} i^5 &= \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{1}{6}n^6 + \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2 \\
\sum_{i=1}^{n} i^6 &= \frac{n(n+1)(2n+1)(3n^4+6n^3-3n+1)}{42} = \frac{1}{7}n^7 + \frac{1}{2}n^6 + \frac{1}{2}n^5 - \frac{1}{6}n^3 + \frac{1}{42}n
\end{aligned}$$

7.32 斯特林数

7.32.1 第一类斯特林数

第一类 Stirling 数 $S(p,k)$ 的一个的组合学解释是：将 p 个物体排成 k 个非空循环排列的方法数。

$S(p,k)$ 的递推公式：$S(p,k)=(p-1)S(p-1,k)+S(p-1,k-1), 1\le k\le p-1$

边界条件：$S(p,0)=0, p\ge 1 \quad S(p,p)=1, p\ge 0$

7.32.2 第二类斯特林数

第二类 Stirling 数 $S(p,k)$ 的一个的组合学解释是：将 p 个物体划分成 k 个非空的不可辨别的（可以理解为盒子没有编号）集合的方法数。

$S(p,k)$ 的递推公式：$S(p,k) = kS(p-1,k) + S(p-1,k-1), 1 \leq k \leq p-1$

边界条件：$S(p,0) = 0, p \geq 1 \quad S(p,p) = 1, p \geq 0$

也有卷积形式：

$$S(n,m) = \frac{1}{m!}\sum_{k=0}^{m}(-1)^k C(m,k)(m-k)^n = \sum_{k=0}^{m}\frac{(-1)^k(m-k)^n}{k!(m-k)!} = \sum_{k=0}^{m}\frac{(-1)^k}{k!}\times\frac{(m-k)^n}{(m-k)!}$$

7.33 各种情况下小球放盒子的方案数

k 个球	m 个盒子	是否允许有空盒子	方案数
各不相同	各不相同	是	m^k
各不相同	各不相同	否	$m!\text{Stirling2}(k,m)$
各不相同	完全相同	是	$\sum_{i=1}^{m}\text{Stirling2}(k,i)$
各不相同	完全相同	否	$\text{Stirling2}(k,m)$
完全相同	各不相同	是	$C(m+k-1,k)$
完全相同	各不相同	否	$C(k-1,m-1)$
完全相同	完全相同	是	$\frac{1}{(1-x)(1-x^2)...(1-x^m)}$ 的 x^k 项的系数
完全相同	完全相同	否	$\frac{x^m}{(1-x)(1-x^2)...(1-x^m)}$ 的 x^k 项的系数

7.34 错排公式

$D_1 = 0, D_2 = 1, D_n = (n-1)(D_{n-2} + D_{n-1})$

7.35 Prufer 编码

对于一棵有 n 个点的带标号的无根树，设 $d[i]$ 为 i 点的度数，那么可以用一个唯一的长度为 $n-2$ 的序列来表示这棵树，其中 i 出现了 $d[i]-1$ 次。若每个点的度数可行的取值是一个集合，则可以通过指数型生成函数来完成计数。

7.36 二项式反演

$$\begin{aligned} f(n) &= \sum_{k=0}^{n} C(n,k)g(k) \\ g(n) &= \sum_{k=0}^{n}(-1)^{n-k}C(n,k)f(k) \end{aligned}$$

7.37 x^k 的转化

$$x^k = \sum_{i=1}^{k} Stirling2(k,i)i!C(x,i)$$

7.38 快速计算素数个数

N 表示要计算的 n 开根号。

```
#include<cstdio>
#include<tr1/unordered_map>
using namespace std::tr1;
const int N=1000000;
int i,j,tot,p[N],f[N];bool v[N];
unordered_map<int,int>T[200];
//the number of natural numbers not greater than m which are
//divisible by no p[i] (1 <= i <= n)
//Phi(m,n)=Phi(m,n-1)-Phi(m/p[n],n-1)
int cal(int m,int n){
  if(!n)return m;
  unordered_map<int,int>::iterator x=T[n].find(m);
  if(x!=T[n].end())return x->second;
  return T[n][m]=cal(m,n-1)-cal(m/p[n-1],n-1);
}
//number of prime less than or equal to n
int Pi(int n){
  if(n<N)return f[n];
  int b=0;
  while(p[b]*p[b]*p[b]<=n)b++;
  int t=cal(n,b)+b-1;
  for(int i=b;p[i]*p[i]<=n;i++)t-=f[n/p[i]]-i;
  return t;
}
int main(){
  for(i=2;i<N;i++){
    if(!v[i])f[p[tot++]=i]=1;
    f[i]+=f[i-1];
    for(j=0;j<tot&&i*p[j]<N;j++){
      v[i*p[j]]=1;
      if(i%p[j]==0)break;
    }
  }
}
```

7.39 Best Theorem

Best Theorem：有向图中以 i 为起点的欧拉回路个数为以 i 为根的树形图个数 $\times$((每个点度数 -1)!)。

Matrix Tree Theorem：以 i 为根的树形图个数 $=$ 基尔霍夫矩阵去掉第 i 行第 i 列的行列式。

从某个点 i 出发并回到 i 的欧拉回路个数 $=$ 以 i 为起点的欧拉回路个数 $\times i$ 的度数。

```
const int N=110,M=200010,P=1000003;
int n,m,i,j,k,x,y,in[N],ou[N],vis[N],g[N][N];ll f[M],a[N][N],ans;
void dfs(int x){
  vis[x]=++m;
  for(int i=1;i<=n;i++)if(g[x][i]&&!vis[i])dfs(i);
}
ll det(int n){
  ll ans=1;bool flag=1;
  for(i=1;i<=n;i++)for(j=1;j<=n;j++)a[i][j]=(a[i][j]%P+P)%P;
  for(i=1;i<=n;i++){
    for(j=i+1;j<=n;j++)while(a[j][i]){
      ll t=a[i][i]/a[j][i];
      for(k=i;k<=n;k++)a[i][k]=(a[i][k]+P-t*a[j][k]%P)%P;
      for(k=i;k<=n;k++)swap(a[i][k],a[j][k]);
      flag^=1;
    }
    ans=ans*a[i][i]%P;
    if(!ans)return 0;
  }
  if(!flag)ans=P-ans;
  return ans;
}
int solve(){
  for(m=0,i=1;i<=n;i++)in[i]=ou[i]=vis[i]=0;
  for(i=0;i<=n;i++)for(j=0;j<=n;j++)a[i][j]=g[i][j]=0;
  int ed=0;
  for(i=1;i<=n;i++)for(read(k);k--;g[i][j]++)read(j),ed++;
  for(dfs(i=1);i<=n;i++)if(!vis[i]&&g[i])return 0;
  for(i=1;i<=n;i++)for(j=1;j<=n;j++)if(g[i][j]){
    x=vis[i],y=vis[j];
    ou[x]+=g[i][j];in[y]+=g[i][j];
    a[x-1][y-1]-=g[i][j],a[x-1][x-1]+=g[i][j];
  }
  for(i=1;i<=m;i++)if(in[i]!=ou[i])return 0;
  if(m==1)return f[g[1][1]];
  ans=det(m-1)*in[1];
  for(i=1;i<=m;i++)ans=ans*f[in[i]-1]%P;
  return ans;
}
int main(){
  for(f[0]=i=1;i<M;i++)f[i]=f[i-1]*i%P;
  while(1){
    read(n);
    if(!n)return 0;
    printf("%d\n",solve());
  }
}
```

7.40 法雷序列

求两分数间分母最小的分数，$1 \leq a,b,c,d \leq 10^9$。

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

```
typedef long long ll;
typedef pair<ll,ll>P;
ll a,b,c,d,t;
ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}
P cal(ll a,ll b,ll c,ll d){
  ll x=a/b+1;
  if(x*d<c)return P(x,1);
  if(!a)return P(1,d/c+1);
  if(a<=b&&c<=d){
    P t=cal(d,c,b,a);
    swap(t.first,t.second);
    return t;
  }
  x=a/b;
  P t=cal(a-b*x,b,c-d*x,d);
  t.first+=t.second*x;
  return t;
}
int main(){
  while(~scanf("%lld%lld%lld%lld",&a,&b,&c,&d)){
    t=gcd(a,b),a/=t,b/=t;
    t=gcd(c,d),c/=t,d/=t;
    P p=cal(a,b,c,d);
    printf("%lld/%lld\n",p.first,p.second);
  }
}
```

7.41 FFT 模任意质数

```
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
const int N=524300,P=1000003,M=1000;
int n,i,j,k,pos[N];
int A[N],B[N],C[N];
namespace FFT{
struct comp{
  long double r,i;comp(long double _r=0,long double _i=0){r=_r;i=_i;}
  comp operator+(const comp x){return comp(r+x.r,i+x.i);}
  comp operator-(const comp x){return comp(r-x.r,i-x.i);}
  comp operator*(const comp x){return comp(r*x.r-i*x.i,r*x.i+i*x.r);}
  comp conj(){return comp(r,-i);}
}A[N],B[N];
int a0[N],b0[N],a1[N],b1[N];
const long double pi=acos(-1.0);
void FFT(comp a[],int n,int t){
  for(int i=1;i<n;i++)if(i<pos[i])swap(a[i],a[pos[i]]);
  for(int d=0;(1<<d)<n;d++){
    int m=1<<d,m2=m<<1;
    long double o=pi*2/m2*t;comp _w(cos(o),sin(o));
    for(int i=0;i<n;i+=m2){
      comp w(1,0);
      for(int j=0;j<m;j++){
        comp&A=a[i+j+m],&B=a[i+j],t=w*A;
```

```
        A=B-t;B=B+t;w=w*_w;
      }
    }
  }
  if(t==-1)for(int i=0;i<n;i++)a[i].r/=n;
}
void mul(int*a,int*b,int*c){//c=a*b
  int i,j;
  for(i=0;i<k;i++)A[i]=comp(a[i],b[i]);
  FFT(A,k,1);
  for(i=0;i<k;i++){
    j=(k-i)&(k-1);
    B[i]=(A[i]*A[i]-(A[j]*A[j]).conj())*comp(0,-0.25);
  }
  FFT(B,k,-1);
  for(i=0;i<k;i++)c[i]=((long long)(B[i].r+0.5))%P;
}
//输入两个多项式，求a*b mod P，保存在c中，c不能为a或b
void mulmod(int*a,int*b,int*c){
  int i;
  for(i=0;i<k;i++)a0[i]=a[i]/M,b0[i]=b[i]/M;
  for(mul(a0,b0,a0),i=0;i<k;i++){
    c[i]=1LL*a0[i]*M*M%P;
    a1[i]=a[i]%M,b1[i]=b[i]%M;
  }
  for(mul(a1,b1,a1),i=0;i<k;i++){
    c[i]=(a1[i]+c[i])%P,a0[i]=(a0[i]+a1[i])%P;
    a1[i]=a[i]/M+a[i]%M,b1[i]=b[i]/M+b[i]%M;
  }
  for(mul(a1,b1,a1),i=0;i<k;i++)c[i]=(1LL*M*(a1[i]-a0[i]+P)+c[i])%P;
}
}
int main(){
  read(n);
  for(k=1;k<n;k<<=1);k<<=1;
  j=__builtin_ctz(k)-1;
  for(i=0;i<k;i++)pos[i]=pos[i>>1]>>1|((i&1)<<j);
  for(i=0;i<n;i++)read(A[i]);
  for(i=0;i<n;i++)read(B[i]);
  FFT::mulmod(A,B,C);
  for(i=0;i<n+n-1;i++)printf("%d ",C[i]);
}
```

8 字符串匹配

8.1 KMP

输入长度为 n 的模式串 a 以及长度为 m 的匹配串 b，下标从 0 开始，依次输出每个匹配成功的位置。

```
int nxt[N];
void kmp(int n,char*a,int m,char*b){
  int i,j;
  for(nxt[0]=j=-1,i=1;i<n;nxt[i++]=j){
    while(~j&&a[j+1]!=a[i])j=nxt[j];
    if(a[j+1]==a[i])j++;
  }
  for(j=-1,i=0;i<m;i++){
    while(~j&&a[j+1]!=b[i])j=nxt[j];
    if(a[j+1]==b[i])j++;
    if(j==n-1)printf("%d ",i),j=nxt[j];
  }
}
```

8.2 最小表示法

```
int n,i,t,a[N<<1];
int minrep(){
  int i=0,j=1,k=0,t;
  while(i<n&&j<n&&k<n)if(t=a[(i+k)%n]-a[(j+k)%n]){
    if(t>0)i+=k+1;else j+=k+1;
    if(i==j)j++;
    k=0;
  }else k++;
  return i<j?i:j;
}
int main(){
  for(scanf("%d",&n);i<n;i++)scanf("%d",&a[i]),a[i+n]=a[i];
  for(t=minrep(),i=0;i<n;i++)printf(i<n-1?"%d ":"%d",a[i+t]);
}
```

8.3 AC 自动机

s 是 t 的后缀等价于 t 串终止节点能通过 fail 指针走到 s 终止节点，即 t 串终止节点是 s 终止节点在 fail 树上的孩子。

```
int tot,son[N][26],id[N],fail[N],q[N];
int n;char s[N];
void insert(int p){
  for(int l=strlen(s),x=0,i=0,w;i<l;i++){
    if(!son[x][w=s[i]-'a'])son[x][w]=++tot;x=son[x][w];
    if(i==l-1)id[x]=p;
  }
}
void make(){
```

```
  int h=1,t=0,i,j,x;fail[0]=-1;
  for(i=0;i<26;i++)if(son[0][i])q[++t]=son[0][i];
  while(h<=t)for(x=q[h++],i=0;i<26;i++)
    if(son[x][i])fail[son[x][i]]=son[fail[x]][i],q[++t]=son[x][i];
    else son[x][i]=son[fail[x]][i];
}
void find(){
  for(int l=strlen(s),x=0,i=0,w,j;i<l;i++){
    x=son[x][w=s[i]-'a'];
    for(j=x;j;j=fail[j])if(id[j])printf("%d ",id[j]);
  }
}
int main(){
  scanf("%d",&n);
  for(int i=1;i<=n;i++)scanf("%s",s),insert(i);
  while(1)scanf("%s",s),find(),puts("");
}
```

8.4 回文串

8.4.1 Manacher

对于一个位置 i，$[i-f[i]+1,i+f[i]-1]$ 是最长的以 i 为中心的奇回文串，$g[i]-i$ 是最长的以 i 为开头的回文串长度。

```
int n,m,i,r,p,f[N],g[N];char a[N],s[N];
int min(int a,int b){return a<b?a:b;}
void up(int&a,int b){if(a<b)a=b;}
int main(){
  while(1){
    scanf("%s",a+1),n=strlen(a+1);
    for(i=1;i<=n;i++)s[i<<1]=a[i],s[i<<1|1]='#';
    s[0]='$',s[1]='#',s[m=(n+1)<<1]='@';
    for(r=p=0,f[1]=1,i=2;i<m;i++){
      for(f[i]=r>i?min(r-i,f[p*2-i]):1;s[i-f[i]]==s[i+f[i]];f[i]++);
      if(i+f[i]>r)r=i+f[i],p=i;
    }
    for(i=0;i<=m;i++)g[i]=0;
    for(i=2;i<m;i++)up(g[i-f[i]+1],i+1);
    for(i=1;i<=m;i++)up(g[i],g[i-1]);
    ans=0;
    for(i=2;i<m;i+=2)printf("%d ",g[i]-i);puts("");
  }
}
```

8.4.2 Palindromic Tree

N：串长。

S：字符集大小。

$text[1..all]$：字符串。

$son[x][y]$：第 x 个点所代表的回文串两边加上字符 y 后的回文串。

$fail[x]$：第 x 个点所代表的回文串的最长回文后缀。

$cnt[x]$：第 x 个点所代表的回文串的出现次数（需建完树后 $count()$ 一遍才可以）。

$len[x]$：第 x 个点所代表的回文串的长度。

```
const int N=300010,S=26;
int all,son[N][S],fail[N],cnt[N],len[N],text[N],last,tot;
int newnode(int l){
  for(int i=0;i<S;i++)son[tot][i]=0;
  cnt[tot]=0,len[tot]=l;
  return tot++;
}
void init(){
  last=tot=all=0;
  newnode(0),newnode(-1);
  text[0]=-1,fail[0]=1;
}
int getfail(int x){
  while(text[all-len[x]-1]!=text[all])x=fail[x];
  return x;
}
void add(int w){
  text[++all]=w;
  int x=getfail(last);
  if(!son[x][w]){
    int y=newnode(len[x]+2);
    fail[y]=son[getfail(fail[x])][w];
    son[x][w]=y;
  }
  cnt[last=son[x][w]]++;
}
void count(){for(int i=tot-1;~i;i--)cnt[fail[i]]+=cnt[i];}
```

8.5 后缀数组

n：串长。

m：字符集大小。

$s[0..n-1]$：字符串。

$sa[1..n]$：字典序第 i 小的是哪个后缀。

$rank[0..n-1]$：后缀 i 的排名。

$height[i]$：$lcp(sa[i], sa[i-1])$。

```
int n,rank[N],sa[N],height[N],tmp[N],cnt[N];char s[N];
void suffixarray(int n,int m){
  int i,j,k;n++;
  for(i=0;i<n*2+5;i++)rank[i]=sa[i]=height[i]=tmp[i]=0;
  for(i=0;i<m;i++)cnt[i]=0;
  for(i=0;i<n;i++)cnt[rank[i]=s[i]]++;
  for(i=1;i<m;i++)cnt[i]+=cnt[i-1];
  for(i=0;i<n;i++)sa[--cnt[rank[i]]]=i;
  for(k=1;k<=n;k<<=1){
    for(i=0;i<n;i++){
      j=sa[i]-k;
      if(j<0)j+=n;
      tmp[cnt[rank[j]]++]=j;
```

```
    }
    sa[tmp[cnt[0]=0]]=j=0;
    for(i=1;i<n;i++){
      if(rank[tmp[i]]!=rank[tmp[i-1]]||rank[tmp[i]+k]!=rank[tmp[i-1]+k])cnt[++j]=i;
      sa[tmp[i]]=j;
    }
    memcpy(rank,sa,n*sizeof(int));
    memcpy(sa,tmp,n*sizeof(int));
    if(j>=n-1)break;
  }
  for(j=rank[height[i=k=0]=0];i<n-1;i++,k++)
    while(~k&&s[i]!=s[sa[j-1]+k])height[j]=k--,j=rank[sa[j]+1];
}
```

8.6 后缀树

在线往右添加字符。

```
const int inf=1<<25,S=256,N=5000;
int root,last,pos,need,remain,acnode,ace,aclen;
struct node{int st,en,lk,son[S];int len(){return min(en,pos+1)-st;}}tree[N<<1];
int n;char text[N],tmp[N];
int new_node(int st,int en=inf){
  node nd;
  nd.st=st;nd.en=en;
  for(int i=nd.lk=0;i<S;i++)nd.son[i]=0;
  tree[++last]=nd;
  return last;
}
char acedge(){return text[ace];}
void addedge(int node){
  if(need)tree[need].lk=node;
  need=node;
}
bool down(int node){
  if(aclen>=tree[node].len()){
    ace+=tree[node].len(),aclen-=tree[node].len(),acnode=node;
    return 1;
  }
  return 0;
}
void init(){
  need=last=remain=ace=aclen=0;
  root=acnode=new_node(pos=-1,-1);
}
void extend(char c){
  text[++pos]=c;need=0;remain++;
  while(remain){
    if(!aclen)ace=pos;
    if(!tree[acnode].son[acedge()]){
      tree[acnode].son[acedge()]=new_node(pos);
      addedge(acnode);
    }else{
      int nxt=tree[acnode].son[acedge()];
      if(down(nxt))continue;
```

```
      if(text[tree[nxt].st+aclen]==c){aclen++;addedge(acnode);break;}
      int split=new_node(tree[nxt].st,tree[nxt].st+aclen);
      tree[acnode].son[acedge()]=split;
      tree[split].son[c]=new_node(pos);
      tree[nxt].st+=aclen;
      tree[split].son[text[tree[nxt].st]]=nxt;
      addedge(split);
    }
    remain--;
    if(acnode==root&&aclen)aclen--,ace=pos-remain+1;
    else acnode=tree[acnode].lk?tree[acnode].lk:root;
  }
}
void show(int x,int dep,int y){
  for(int i=0;i<dep;i++)putchar('-');
  for(int i=tree[x].st;i<min(tree[x].en,pos+1);i++)printf("%c",text[i]);puts(":");
  for(int i=0;i<S;i++)if(tree[x].son[i])show(tree[x].son[i],dep+2,i);
}
int search(){
  scanf("%s",tmp+1);
  n=strlen(tmp+1);
  int x=root,i=1,j;
  while(i<=n){
    if(tree[x].son[tmp[i]]){
      x=tree[x].son[tmp[i]];
      j=tree[x].st;
      while(i<=n&&j<min(tree[x].en,pos+1))if(tmp[i]==text[j])i++,j++;else return 0;
    }else return 0;
  }
  return 1;
}
int main(){
  init();
  scanf("%s",tmp+1);
  n=strlen(tmp+1);
  for(int i=1;i<=n;i++)extend(tmp[i]);extend('$');
  show(root,0,0);
  while(1)printf("%d\n",search());
}
```

8.7 后缀自动机

在线往右添加字符。

```
struct SuffixAutomaton{
  enum{N_CHAR=26,MX_LEN=1100000};
  struct Node{Node *fail,*next[N_CHAR];int val,right;};
  Node mempool[MX_LEN*2];int n_node;
  Node*new_node(int v){
    Node*p=&mempool[n_node++];
    for(int i=0;i<N_CHAR;++i)p->next[i]=0;
    return p->fail=0,p->right=0,p->val=v,p;
  }
  Node*root,*last;
  SuffixAutomaton(){clear();}
```

```
  void clear(){root=last=new_node(n_node=0);}
  void add(int c){
    Node*p=last,*np=new_node(p->val+1);
    while(p&&!p->next[c])p->next[c] = np,p = p->fail;
    if(!p)np->fail=root;
    else{
      Node*q=p->next[c];
      if(p->val+1==q->val)np->fail=q;
      else{
        Node*nq=new_node(p->val+1);
        for(int i=0;i<N_CHAR;++i)nq->next[i]=q->next[i];
        nq->fail=q->fail,q->fail=np->fail=nq;
        while(p&&p->next[c]==q)p->next[c]=nq,p=p->fail;
      }
    }
    last=np,np->right=1;
  }
  Node*go(const char*s){
    int cL=0;//与s匹配的长度
    Node*p=root;
    for(int i=0;s[i];++i){
      int c=s[i]-'a';
      if(p->next[c])p=p->next[c],++cL;
      else{
        while(p&&!p->next[c])p=p->fail;
        if(!p)cL=0,p=root;else cL=p->val+1,p=p->next[c];
      }
    }
    return p;
  }
  int d[MX_LEN*2];Node*b[MX_LEN*2];
  void topological_sort(){
    for(int i=0;i<=n_node;++i)d[i]=0;
    int mx_val=0;
    for(int i=0;i<n_node;++i)mx_val=max(mx_val,mempool[i].val),d[mempool[i].val]++;
    for(int i=1;i<=mx_val;++i)d[i]+=d[i-1];
    for(int i=0;i<n_node;++i)b[--d[mempool[i].val]]=&mempool[i];
  }
  void update_right(){
    topological_sort();
    for(int i=n_node-1;i;--i)b[i]->fail->right+=b[i]->right;
  }
};
```

8.8 后缀自动机 - Claris

```
int tot=1,last=1,pre[N<<1],son[N<<1][S],ml[N<<1];
void extend(int w){
  int p=++tot,x=last,r,q;
  for(ml[last=p]=ml[x]+1;x&&!son[x][w];x=pre[x])son[x][w]=p;
  if(!x)pre[p]=1;
  else if(ml[x]+1==ml[q=son[x][w]])pre[p]=q;
  else{
    pre[r=++tot]=pre[q];memcpy(son[r],son[q],sizeof son[r]);
    ml[r]=ml[x]+1;pre[p]=pre[q]=r;
```

```
    for(;x&&son[x][w]==q;x=pre[x])son[x][w]=r;
  }
}
```

8.9 后缀平衡树

在线往左添加字符，一个串 S 的出现次数 = 字典序小于 $S*$ 的后缀个数 − 字典序小于 S 的后缀个数，其中 $*$ 为字符集中没出现的字符，且比任意字符都要大。

len：串长。

$s[i]$：从右往左第 i 个字符。

比较从右往左第 i 个字符开始的后缀与从右往左第 j 个字符开始的后缀的字典序等价于比较 $tm[i]$ 与 $tm[j]$。

$ins(len)$：插入从右往左第 len 个字符开始的后缀。

```
typedef unsigned long long ll;
const ll inf=1ULL<<63;
const double A=0.8;
ll tl[N],tr[N],tm[N];
int size[N],son[N][2],f[N],v[N],tot,root,id[N],cnt;
char s[N];
bool cmp(int a,int b){return s[a]==s[b]?tm[a-1]>tm[b-1]:s[a]>s[b];}
int ins(int x,int p){
  int b=cmp(p,v[x]);
  if(!son[x][b]){
    son[x][b]=++tot;f[tot]=x;v[tot]=p;
    if(!b)tl[tot]=tl[x],tr[tot]=tm[x];else tl[tot]=tm[x],tr[tot]=tr[x];
    tm[tot]=(tl[tot]+tr[tot])>>1;
    return tot;
  }else return ins(son[x][b],p);
}
void dfs(int x){
  if(son[x][0])dfs(son[x][0]);
  id[++cnt]=x;
  if(son[x][1])dfs(son[x][1]);
}
int build(int fa,int l,int r,ll a,ll b){
  int mid=(l+r)>>1,x=id[mid];
  f[x]=fa;son[x][0]=son[x][1]=0;size[x]=1;tl[x]=a;tr[x]=b;tm[x]=(a+b)>>1;
  if(l==r)return x;
  if(l<mid)size[x]+=size[son[x][0]=build(x,l,mid-1,a,tm[x])];
  if(r>mid)size[x]+=size[son[x][1]=build(x,mid+1,r,tm[x],b)];
  return x;
}
int rebuild(int x){cnt=0;dfs(x);return build(f[x],1,cnt,tl[x],tr[x]);}
void insert(int p){
  if(!root){root=tot=size[1]=1;v[1]=p;tr[1]=inf,tm[1]=inf>>1;return;}
  int x=ins(root,p);
  int deep=0,z=x;while(z)size[z]++,z=f[z],deep++;
  if(deep<log(tot)/log(1/A))return;
  while(1.0*size[son[x][0]]<A*size[x]&&1.0*size[son[x][1]]<A*size[x])x=f[x];
  if(!x)return;
  if(x==root){root=rebuild(x);return;}
  int y=f[x],b=son[y][1]==x,now=rebuild(x);
```

```
  son[y][b]=now;
}
```

9 随机化

9.1 Pollard Rho

```
#include<cstdio>
#include<algorithm>
#define C 2730
#define S 3
using namespace std;
typedef long long ll;
ll n;
ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}
ll mul(ll a,ll b,ll n){return(a*b-(ll)(a/(long double)n*b+1e-3)*n+n)%n;}
ll pow(ll a, ll b, ll n){
  ll d=1;
  a%=n;
  while(b){
    if(b&1)d=mul(d,a,n);
    a=mul(a,a,n);
    b>>=1;
  }
  return d;
}
bool check(ll a,ll n){
  ll m=n-1,x,y;int i,j=0;
  while(!(m&1))m>>=1,j++;
  x=pow(a,m,n);
  for(i=1;i<=j;x=y,i++){
    y=pow(x,2,n);
    if((y==1)&&(x!=1)&&(x!=n-1))return 1;
  }
  return y!=1;
}
bool miller_rabin(int times,ll n){
  ll a;
  if(n==1)return 0;
  if(n==2)return 1;
  if(!(n&1))return 0;
  while(times--)if(check(rand()%(n-1)+1,n))return 0;
  return 1;
}
ll pollard_rho(ll n,int c){
  ll i=1,k=2,x=rand()%n,y=x,d;
  while(1){
    i++,x=(mul(x,x,n)+c)%n,d=gcd(y-x,n);
    if(d>1&&d<n)return d;
    if(y==x)return n;
    if(i==k)y=x,k<<=1;
  }
}
void findfac(ll n,int c){
  if(n==1)return;
  if(miller_rabin(S,n)){
    //找到了质因子n
    return;
  }
```

```
  ll m=n;
  while(m==n)m=pollard_rho(n,c--);
  findfac(m,c),findfac(n/m,c);
}
int main(){while(~scanf("%lld",&n))findfac(n,C);}
```

9.2 最小圆覆盖

给定 n 个点 $b[0], b[1], ..., b[n-1]$，返回最小的能覆盖所有点的圆，圆心为 O，半径为 R。

```
double R,eps=1e-10;
struct P{double x,y;}a[N],O;
double dis(P x,P y){return sqrt((x.x-y.x)*(x.x-y.x)+(x.y-y.y)*(x.y-y.y));}
P center(P x,P y,P z){
  double a1=y.x-x.x,b1=y.y-x.y,
         c1=(a1*a1+b1*b1)/2,a2=z.x-x.x,
         b2=z.y-x.y,c2=(a2*a2+b2*b2)/2,
         d=a1*b2-a2*b1;
  return (P){x.x+(c1*b2-c2*b1)/d,x.y+(a1*c2-a2*c1)/d};
}
void cal(int n,P*b){
  int i,j,k,n=0;
  for(i=0;i<n;i++)a[i]=b[i];
  for(i=0;i<n;i++)swap(a[rand()%n],a[i]);
  for(O=a[0],R=0,i=1;i<n;i++)if(dis(a[i],O)>R+eps)
    for(O=a[i],R=0,j=0;j<i;j++)if(dis(a[j],O)>R+eps){
      O=(P){(a[i].x+a[j].x)/2,(a[i].y+a[j].y)/2},R=dis(O,a[i]);
      for(k=0;k<j;k++)if(dis(a[k],O)>R+eps)O=center(a[k],a[j],a[i]),R=dis(O,a[i]);
    }
}
```

10 计算几何

10.1 半平面交

```
const int N=600;
const double eps=1e-10;
struct P{
  double x,y;
  P(){x=y=0;}
  P(double _x,double _y){x=_x,y=_y;}
  P operator-(const P&a)const{return P(x-a.x,y-a.y);}
  P operator+(const P&a)const{return P(x+a.x,y+a.y);}
  P operator*(double a)const{return P(x*a,y*a);}
  void read(){scanf("%lf%lf",&x,&y);}
}p[N],a[N];
struct L{
  P p,v;double a;
  L(){}
  L(P _p,P _v){p=_p,v=_v;}
  bool operator<(const L&b)const{return a<b.a;}
  void cal(){a=atan2(v.y,v.x);}
}line[N],q[N];
int n,m,i,cl;
double cross(const P&a,const P&b){return a.x*b.y-a.y*b.x;}
//新的半平面，在这条向量a->b的逆时针方向
void newL(const P&a,const P&b){line[++cl]=L(a,b-a);}
bool left(const P&p,const L&l){return cross(l.v,p-l.p)>0;}
P pos(const L&a,const L&b){
  P x=a.p-b.p;double t=cross(b.v,x)/cross(a.v,b.v);
  return a.p+a.v*t;
}
double halfplane(){
  for(int i=1;i<=cl;i++)line[i].cal();
  sort(line+1,line+cl+1);
  int h=1,t=1;
  q[1]=line[1];
  for(int i=2;i<=cl;i++){
    while(h<t&&!left(p[t-1],line[i]))t--;
    while(h<t&&!left(p[h],line[i]))h++;
    if(fabs(cross(q[t].v,line[i].v))<eps)q[t]=left(q[t].p,line[i])?q[t]:line[i];
    else q[++t]=line[i];
    if(h<t)p[t-1]=pos(q[t],q[t-1]);
  }
  while(h<t&&!left(p[t-1],q[h]))t--;
  p[t]=pos(q[t],q[h]);
  if(t-h<=1)return 0;
  double ans=0;
  for(int i=h;i<t;i++)ans+=cross(p[i],p[i+1]);
  return (ans+cross(p[t],p[h]))/2;
}
int main(){
  scanf("%d",&n);
  while(n--){
    scanf("%d",&m);
    for(i=0;i<m;i++)a[i].read();
    for(i=0;i<m;i++)newL(a[i],a[(i+1)%m]);
```

```
  }
  printf("%.3f",halfplane());
}
```

10.2 最小矩形覆盖

求出凸包后旋转卡壳。

```
typedef double D;
struct P{D x,y;P(){}P(D _x,D _y){x=_x,y=_y;}}p[N],pp[N],hull[N],pivot,A,B,C,rect[8];
int n,i,j,l,r,k;
D w,h,ans=1e20,tmp,len;
bool del[N];
int zero(D x){return fabs(x)<1e-4;}
int sig(D x){if(fabs(x)<1e-8)return 0;return x>0?1:-1;}
D cross(P A,P B,P C){return(B.x-A.x)*(C.y-A.y)-(B.y-A.y)*(C.x-A.x);}
D distsqr(P A,P B){return(A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y);}
bool cmp(P a,P b){
  D t=cross(pivot,a,b);
  return sig(t)==1||sig(t)==0&&sig(distsqr(pivot,a)-distsqr(pivot,b))==-1;
}
void convexhull(int n,P stck[],int&m){
  int i,k,top;
  for(i=0;i<n;i++)pp[i]=p[i];
  for(k=0,i=1;i<n;i++)if(pp[i].y<pp[k].y||(pp[i].y==pp[k].y&&pp[i].x<pp[k].x))k=i;
  pivot=pp[k];pp[k]=pp[0];pp[0]=pivot;
  sort(pp+1,pp+n,cmp);
  stck[0]=pp[0];stck[1]=pp[1];
  for(top=1,i=2;i<n;i++){
    while(top&&sig(cross(pp[i],stck[top],stck[top-1]))>=0)--top;
    stck[++top]=pp[i];
  }
  m=top+1;
}
D area(P A,P B,P C){return fabs(cross(A,B,C));}
P vertical(P A,P B){return P(A.x-B.y+A.y,A.y+B.x-A.x);}
int main(){
  scanf("%d",&n);
  for(i=0;i<n;i++)scanf("%lf%lf",&p[i].x,&p[i].y);
  convexhull(n,hull,n);
  for(i=1;i<n;i++)if(zero(hull[i].x-hull[i-1].x)&&zero(hull[i].y-hull[i-1].y))del[i]=1;
  for(k=i=0;i<n;i++)if(!del[i])hull[k++]=hull[i];
  for(hull[n=k]=hull[i=0];i<n;i++){
    A=hull[i],B=hull[i+1],C=vertical(A,B);
    while(sig(area(A,B,hull[j])-area(A,B,hull[j+1]))<1)j=(j+1)%n;
    while(sig(cross(A,C,hull[l])-cross(A,C,hull[l+1]))<1)l=(l+1)%n;
    while(sig(cross(A,C,hull[r])-cross(A,C,hull[r+1]))>-1)r=(r+1)%n;
    len=sqrt(distsqr(A,B));
    h=area(A,B,hull[j])/len;
    w=(cross(A,C,hull[l])-cross(A,C,hull[r]))/len;
    if(sig(h*w-ans)==-1){
      ans=h*w;
      tmp=area(A,B,hull[l])/len/len;
      rect[0]=P(hull[l].x+tmp*(A.x-C.x),hull[l].y+tmp*(A.y-C.y));
      tmp=h/len;
```

```
      rect[3]=P(rect[0].x+tmp*(C.x-A.x),rect[0].y+tmp*(C.y-A.y));
      tmp=w/len;
      rect[1]=P(rect[0].x+tmp*(B.x-A.x),rect[0].y+tmp*(B.y-A.y));
      rect[2]=P(rect[3].x+tmp*(B.x-A.x),rect[3].y+tmp*(B.y-A.y));
    }
  }
  for(i=0;i<4;i++)rect[i+4]=rect[i];
  for(j=0,i=1;i<4;i++)
    if(sig(rect[i].y-rect[j].y)==-1||
       sig(rect[i].y-rect[j].y)==0&&sig(rect[i].x-rect[j].x)==-1)j=i;
  printf("%.0f.00000\n",ans);
  for(i=0;i<4;i++)printf("%.0f.00000 %.0f.00000\n",rect[j+i].x,rect[j+i].y);
}
```

10.3 三维凸包

```
#define PR 1e-8
#define N 620
struct TPoint{
  double x,y,z;
  TPoint(){}
  TPoint(double _x,double _y,double _z):x(_x),y(_y),z(_z){}
  TPoint operator+(const TPoint p){return TPoint(x+p.x,y+p.y,z+p.z);}
  TPoint operator-(const TPoint p){return TPoint(x-p.x,y-p.y,z-p.z);}
  TPoint operator*(const TPoint p){//叉积
    return TPoint(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);
  }
  TPoint operator*(double p){return TPoint(x*p,y*p,z*p);}
  TPoint operator/(double p){return TPoint(x/p,y/p,z/p);}
  double operator^(const TPoint p){return x*p.x+y*p.y+z*p.z;}//点积
}center;
struct fac{
  int a,b,c;//凸包一个面上的三个点的编号
  bool ok;//该面是否是最终凸包中的面
};
struct T3dhull{
  int n;//初始点数
  TPoint ply[N];//初始点
  int trianglecnt;//凸包上三角形数
  fac tri[N];//凸包三角形
  int vis[N][N];//点i到点j是属于哪个面
  double dist(TPoint a){//两点长度
    return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
  }
  double area(TPoint a,TPoint b,TPoint c){//三角形面积*2
    return dist((b-a)*(c-a));
  }
  double volume(TPoint a,TPoint b,TPoint c,TPoint d){//四面体有向体积*6
    return (b-a)*(c-a)^(d-a);
  }
  double ptoplane(TPoint &p,fac &f){//正：点在面同向
    TPoint m=ply[f.b]-ply[f.a],n=ply[f.c]-ply[f.a],t=p-ply[f.a];
    return (m*n)^t;
  }
  void deal(int p,int a,int b){
```

```
    int f=vis[a][b];
    fac add;
    if(tri[f].ok){
      if((ptoplane(ply[p],tri[f]))>PR)dfs(p,f);else{
        add.a=b,add.b=a,add.c=p,add.ok=1;
        vis[p][b]=vis[a][p]=vis[b][a]=trianglecnt;
        tri[trianglecnt++]=add;
      }
    }
  }
  void dfs(int p,int cnt){//维护凸包，如果点p 在凸包外更新凸包
    tri[cnt].ok=0;
    deal(p,tri[cnt].b,tri[cnt].a);
    deal(p,tri[cnt].c,tri[cnt].b);
    deal(p,tri[cnt].a,tri[cnt].c);
  }
  bool same(int s,int e){//判断两个面是否为同一面
    TPoint a=ply[tri[s].a],b=ply[tri[s].b],c=ply[tri[s].c];
    return fabs(volume(a,b,c,ply[tri[e].a]))<PR
    &&fabs(volume(a,b,c,ply[tri[e].b]))<PR
    &&fabs(volume(a,b,c,ply[tri[e].c]))<PR;
  }
  void construct(){//构建凸包
    int i,j;
    trianglecnt=0;
    if(n<4)return;
    bool tmp=1;
    for(i=1;i<n;i++){//前两点不共点
      if((dist(ply[0]-ply[i]))>PR){
        swap(ply[1],ply[i]);tmp=0;break;
      }
    }
    if(tmp)return;
    tmp=1;
    for(i=2;i<n;i++){//前三点不共线
      if((dist((ply[0]-ply[1])*(ply[1]-ply[i])))>PR){
        swap(ply[2],ply[i]); tmp=0; break;
      }
    }
    if(tmp)return;
    tmp=1;
    for(i=3;i<n;i++)//前四点不共面
      if(fabs((ply[0]-ply[1])*(ply[1]-ply[2])^(ply[0]-ply[i]))>PR){
        swap(ply[3],ply[i]);tmp=0;break;
      }
    if(tmp)return;
    fac add;
    for(i=0;i<4;i++){//构建初始四面体
      add.a=(i+1)%4,add.b=(i+2)%4,add.c=(i+3)%4,add.ok=1;
      if((ptoplane(ply[i],add))>0) swap(add.b,add.c);
      vis[add.a][add.b]=vis[add.b][add.c]=vis[add.c][add.a]=trianglecnt;
      tri[trianglecnt++]=add;
    }
    for(i=4;i<n;i++){//构建更新凸包
      for(j=0;j<trianglecnt;j++)
        if(tri[j].ok&&(ptoplane(ply[i],tri[j]))>PR){
          dfs(i,j);break;
```

```
        }
      }
      int cnt=trianglecnt;
      trianglecnt=0;
      for(i=0;i<cnt;i++)
        if(tri[i].ok)
          tri[trianglecnt++]=tri[i];
    }
    double area(){//表面积
      double ret=0;
      for(int i=0;i<trianglecnt;i++)ret+=area(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
      return ret/2.0;
    }
    double volume(){//体积
      TPoint p(0,0,0);
      double ret=0;
      for(int i=0;i<trianglecnt;i++)
        ret+=volume(p,ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
      return fabs(ret/6);
    }
    int facetri(){return trianglecnt;}//表面三角形数
    int facepolygon(){//表面多边形数
      int ans=0,i,j,k;
      for(i=0;i<trianglecnt;i++){
        for(j=0,k=1;j<i;j++){
          if(same(i,j)){k=0;break;}
        }
        ans+=k;
      }
      return ans;
    }
    TPoint barycenter(){//重心
      TPoint ans(0,0,0),o(0,0,0);
      double all=0;
      for(int i=0;i<trianglecnt;i++){
        double vol=volume(o,ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]);
        ans=ans+(o+ply[tri[i].a]+ply[tri[i].b]+ply[tri[i].c])/4.0*vol;
        all+=vol;
      }
      return ans/all;
    }
    double ptoface(TPoint p,int i){//点到面的距离
      return fabs(volume(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c],p)
                  /area(ply[tri[i].a],ply[tri[i].b],ply[tri[i].c]));
    }
}a;
int main(){
  scanf("%d",&a.n);
  for(int i=0;i<a.n;i++)scanf("%lf%lf%lf",&a.ply[i].x,&a.ply[i].y,&a.ply[i].z);
  a.construct();
  center=a.barycenter();
  double tmp=1e15;
  for(int i=0;i<a.trianglecnt;i++)tmp=min(tmp,a.ptoface(center,i));
  printf("%.6f",tmp);
}
```

10.4 球缺

半径为 r，高度为 h 的球缺的体积为 $\frac{h^2(3r-h)\pi}{3}$。

10.5 计算几何模板大全

```
#include<cstdio>
#include<algorithm>
#include<cmath>
using namespace std;
double eps=1e-9;
//------------------------------------------------------------
//       Fundamental
//------------------------------------------------------------
int sgn(double x) {
  if( x < -eps ) return -1;
  if( x >  eps ) return  1;
  return 0;
}
//二次方程
bool Quadratic(double A, double B, double C, double *t0, double *t1) {
    double discrim = B * B - 4.f * A * C;
    if (discrim < 0.) return false;
    double rootDiscrim = sqrt(discrim);
    double q;
    if (B < 0) q = -.5f * (B - rootDiscrim);
    else       q = -.5f * (B + rootDiscrim);
    *t0 = q / A;
    *t1 = C / q;
    if (*t0 > *t1) swap(*t0, *t1);
    return true;
}
struct vec {
  double x,y;
  vec(){x=y=0;}
  vec(double _x,double _y){x=_x,y=_y;}

  vec   operator + (vec v) {return vec(x+v.x,y+v.y);}
  vec   operator - (vec v) {return vec(x-v.x,y-v.y);}
  vec   operator * (double v) {return vec(x*v,y*v);}
  vec   operator / (double v) {return vec(x/v,y/v);}

  double operator * (vec v) {return x*v.x + y*v.y;}

  double len()     {return hypot(x,y); }
  double len_sqr() {return x*x + y*y; }

  //逆时针旋转
  vec rotate(double c) {return vec(x*cos(c)-y*sin(c),x*sin(c)+y*cos(c));}
  vec trunc (double l) {return (*this) * l / len();}
  vec rot90 () {return vec(-y,x);}
};
double cross(vec a,vec b) {return a.x*b.y - a.y*b.x;}
//计算 a,b 间的角度
double get_angle(vec a,vec b) {return fabs(atan2(fabs(cross(a,b)),a*b));}
```

```
//直线插值
vec lerp(vec a,vec b,double t) {return a * (1-t) + b * t;}

//判断点是否在线段上(包含端点)
bool point_on_segment(vec p,vec a,vec b) {
  return sgn( cross(b-a,p-a) ) == 0 && sgn( (p-a)*(p-b) ) <= 0;
}

//判断线段ab,pq间是否有交点
int has_intersection(vec a,vec b,vec p,vec q) {
  int d1 = sgn( cross(b-a,p-a) ),d2 = sgn( cross(b-a,q-a) );
  int d3 = sgn( cross(q-p,a-p) ),d4 = sgn( cross(q-p,b-p) );
  if( d1 * d2 < 0 && d3 * d4 < 0 )
    return  1; //有交点，且交点不在端点
  if( ( d1 == 0 && point_on_segment(p,a,b) )||
    ( d2 == 0 && point_on_segment(q,a,b) )||
    ( d3 == 0 && point_on_segment(a,p,q) )||
    ( d4 == 0 && point_on_segment(b,p,q) ))
    return -1; //重合或交点在端点上
  return 0;
}

//直线求交点，需保证p!=q,a!=b
int line_intersection(vec a,vec b,vec p,vec q,vec &o,double *t=0) {
  double U = cross(p-a,q-p);
  double D = cross(b-a,q-p);
  if( sgn(D) == 0 )
    return sgn(U)==0 ? 2:0;//重叠|平行
  o = a + (b-a) * (U/D);
  if(t) *t = U/D;
  return 1;
}

//点p到直线ab距离
double dist_point_to_line(vec p,vec a,vec b) {
  return fabs(cross(p-a,b-a))/(b-a).len();
}

//点p到线段ab距离
double dist_point_to_segment(vec p,vec a,vec b) {
  if( sgn( (p-a)*(b-a) ) >= 0 && sgn( (p-b)*(a-b) ) >= 0 )
    return fabs(cross(p-a,b-a))/(b-a).len();
  return min( (p-a).len() , (p-b).len() );
}

//-----------------------------------------------------------
//       Circle
//-----------------------------------------------------------
struct circle {
  vec c;double r;
  circle(){c=vec(0,0),r=0;}
  circle(vec _c,double _r){c=_c,r=_r;}
};

//圆直线交点，交点是lerp(a,b,*t0)和lerp(a,b,*t1)
bool circle_line_intersection(circle c,vec a,vec b,double *t0,double *t1) {
  vec d = b - a;
```

```
  double A = d * d;
  double B = d * (a-c.c) * 2.0;
  double C = (a-c.c).len_sqr() - c.r * c.r;
  return Quadratic(A,B,C,t0,t1);
}

//圆圆相交
bool circle_circle_intersection(circle a,circle b,vec &p1,vec &p2) {
  double d = (a.c-b.c).len();
  if( d > a.r + b.r || d < fabs(a.r-b.r) ) return false;//相离|内含
  double l = ( (a.c-b.c).len_sqr() + a.r*a.r - b.r*b.r ) / (2*d);
  double h = sqrt(a.r*a.r-l*l);
  vec vl = (b.c-a.c).trunc(l),vh = vl.rot90().trunc(h);
  p1 = a.c + vl + vh;
  p2 = a.c + vl - vh;
  return true;
}

//圆和三角形abo交的面积，o是圆心
double circle_triangle_intersection_area(circle c,vec a,vec b) {
  if( sgn(cross(a-c.c,b-c.c)) == 0 ) return 0;
  vec q[5];
  int len = 0;
  double t0,t1;
  q[len++] = a;
  if( circle_line_intersection(c,a,b,&t0,&t1) ) {
    if( 0 <= t0 && t0 <= 1 ) q[len++] = lerp(a,b,t0);
    if( 0 <= t1 && t1 <= 1 ) q[len++] = lerp(a,b,t1);
  }
  q[len++] = b;
  double s = 0;
  for(int i=1;i<len;++i) {
    vec z = (q[i-1] + q[i])/2;
    if( (z-c.c).len_sqr() <= c.r*c.r )
      s += fabs( cross(q[i-1]-c.c,q[i]-c.c) ) / 2;
    else
      s += c.r*c.r*get_angle(q[i-1]-c.c,q[i]-c.c) / 2;
  }
  return s;
}

//------------------------------------------------------------
//       Polygon
//------------------------------------------------------------
//多边形与圆交的面积
double circle_polygon_intersection_area(circle c,vec *v,int n) {
  double s = 0;
  for(int i=0;i<n;++i) {
    int j = (i+1) % n;
    s += circle_triangle_intersection_area(c,v[i],v[j])
       * sgn( cross(v[i]-c.c,v[j]-c.c) );
  }
  return fabs(s);
}

//切割多边形
//顶点按逆时针给，保留有向直线a->b左侧的部分
```

```
int polygon_cut(vec *v,int n,vec a,vec &b,vec *o) {
  int len = 0;
  for(int i=0;i<n;++i) {
    if( cross(v[i]-a,b-a) <= 0 ) o[len++] = v[i];
    vec c;double t;
    if( line_intersection(v[i],v[(i+1)%n],a,b,c,&t) && t > 0 && t < 1 )
      o[len++] = c;
  }
  return len;
}

//判点在是否在多边形内
bool point_in_polygon(vec *v,int n,vec c) {
  double s = 0;
  for(int i=0;i<n;++i) {
    vec a = v[i] - c,b = v[(i+1)%n] - c;
    double ang = acos( a*b/(a.len() * b.len()) );
    s += cross(a,b) < 0 ? ang : -ang;
  }
  return sgn(s) != 0;// s=0在多边形外,s=pi在边缘上,否则在多边形内
}

//凸包, 不可有重复点
bool cmpXY(vec a,vec b) {
  if( sgn(a.x-b.x) ) return a.x < b.x;
  return a.y < b.y;
}
int convex_hull(vec* v,int n,vec *z) {
  sort(v,v+n,cmpXY);
  int m = 0;
  for(int i=0;i<n;++i) {
    while( m > 1 && cross(z[m-1]-z[m-2],v[i]-z[m-2]) <= 0 ) --m;
    z[m++] = v[i];
  }
  int k = m;
  for(int i=n-2;i>=0;--i) {
    while( m > k && cross(z[m-1]-z[m-2],v[i]-z[m-2]) <= 0 ) --m;
    z[m++] = v[i];
  }
  if( n > 1 ) --m;
  return m;
}

//-------------------------------------------------------------
//        Misc
//-------------------------------------------------------------
//绕轴旋转矩阵,使用列向量,matrix::I()是单位阵
//注意：对应法线的变换矩阵是Inverse(Transpose(R))
//verified HDU 5388
matrix rotate(double x,double y,double z,double d) {
    double len = sqrt( x*x + y*y + z*z );
    x/=len;y/=len;z/=len;
    matrix K;
    K.v[0][1] = -z;K.v[0][2] =  y;
    K.v[1][0] =  z;K.v[1][2] = -x;
    K.v[2][0] = -y;K.v[2][1] =  x;
    return matrix::I() + K * sin(d) + K * K * (1 - cos(d));
```

```
}

//三角形面积，海伦公式，a,b,c为三边长
double get_triangle_area(double a,double b,double c) {
  double s = (a+b+c) / 2;
  return sqrt( s*(s-a)*(s-b)*(s-c) );
}
```

10.6 曼哈顿凸包

先输出周长再输出面积。

```
#include<cstdio>
#include<algorithm>
using namespace std;
int n,i,r,t,q[100010],A,B,C,D;long long ans;struct P{int x,y;}a[100010];
bool cmp(P a,P b){return a.x==b.x?a.y<b.y:a.x<b.x;}
int main(){
  A=C=~0U>>1,B=D=-A;
  for(scanf("%d",&n),i=1;i<=n;i++){
    scanf("%d%d",&a[i].x,&a[i].y);
    A=min(A,a[i].x);
    B=max(B,a[i].x);
    C=min(C,a[i].y);
    D=max(D,a[i].y);
  }
  sort(a+1,a+n+1,cmp);
  for(i=1;i<=n;i++)if(!t||a[i].y>r)r=a[i].y,q[++t]=i;
  for(i=q[r=t]+1;i<=n;q[++t]=i++)while(t>r&&a[i].y>=a[q[t]].y)t--;
  for(i=2;i<=t;i++)ans+=1LL*min(a[q[i-1]].y,a[q[i]].y)*(a[q[i]].x-a[q[i-1]].x);
  for(t=0,i=1;i<=n;i++)if(!t||a[i].y<r)r=a[i].y,q[++t]=i;
  for(i=q[r=t]+1,i=q[t]+1;i<=n;q[++t]=i++)while(t>r&&a[i].y<=a[q[t]].y)t--;
  for(i=2;i<=t;i++)ans-=1LL*max(a[q[i-1]].y,a[q[i]].y)*(a[q[i]].x-a[q[i-1]].x);
  printf("%lld\n%lld",2LL*B+2LL*D-2LL*A-2LL*C,ans);
}
```

10.7 圆的面积并

圆并算法，时间复杂度 $O(n^2 \log n)$。

```
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
typedef pair<double,double>P;
const int N=1010;
const double PI=acos(-1.0),eps=1e-8;
int n,i,j,del[N],t;
double a[N],b[N],r[N],d,x,y,u,ans;
P q[N],tmp;
double sqr(double x){return x*x;}
double dis(double x1,double y1,double x2,double y2){
  return sqrt(sqr(x1-x2)+sqr(y1-y2));
}
```

```
double angle(double a,double b,double c){
  return acos((sqr(a)+sqr(b)-sqr(c))/(2*a*b));
}
int sig(double x){
  if(fabs(x)<eps)return 0;
  return x>0?1:-1;
}
void cal(int i,double L,double R){
  double x1=a[i]+r[i]*cos(L),y1=b[i]+r[i]*sin(L),
         x2=a[i]+r[i]*cos(R),y2=b[i]+r[i]*sin(R);
  ans+=r[i]*r[i]*(R-L-sin(R-L))+x1*y2-x2*y1;
}
int main(){
  scanf("%d",&n);
  for(i=0;i<n;i++){
    scanf("%lf%lf%lf",&a[i],&b[i],&r[i]);
    for(j=0;j<i;j++)if(!sig(r[i]-r[j])&&!sig(a[i]-a[j])&&!sig(b[i]-b[j])){
      r[i]=0;
      break;
    }
  }
  for(i=0;i<n;i++)for(j=0;j<n;j++)
    if(i!=j&&sig(r[j]-r[i])>=0&&sig(dis(a[i],b[i],a[j],b[j])-r[j]+r[i])<=0){
      del[i]=1;
      break;
    }
  for(i=0;i<n;i++)if(sig(r[i])&&!del[i]){
    for(t=j=0;j<n;j++)if(i!=j){
      d=dis(a[i],b[i],a[j],b[j]);
      if(sig(d-r[i]-r[j])>=0||sig(d-fabs(r[i]-r[j]))<=0)continue;
      x=atan2(b[j]-b[i],a[j]-a[i]),y=angle(r[i],d,r[j]);
      tmp=P(x-y,x+y);
      if(sig(tmp.first)<=0&&sig(tmp.second)<=0)q[t++]=P(2*PI+tmp.first,2*PI+tmp.second);
      else if(sig(tmp.first)<0)q[t++]=P(2*PI+tmp.first,2*PI),q[t++]=P(0,tmp.second);
      else q[t++]=tmp;
    }
    if(t)sort(q,q+t);
    for(u=0,j=0;j<t;j++)
      if(sig(u-q[j].first)>=0)u=max(u,q[j].second);
      else cal(i,u,q[j].first),u=q[j].second;
    if(!sig(u))ans+=r[i]*r[i]*2*PI;else cal(i,u,2*PI);
  }
  ans/=2;
  printf("%.3f",ans);
}
```

10.8 平面图

给定一张平面图，不保证连通，以及若干个点，进行平面图求域以及点定位，时间复杂度 $O(n \log n)$。

cnt 表示封闭区域个数，无限域编号为 0，$from[i]$ 表示第 i 条边所属区域，$id[i]$ 表示第 i 个询问点所属区域。

```
#include<cstdio>
```

```
#include<cmath>
#include<set>
#include<map>
#include<algorithm>
using namespace std;
const double eps=1e-8;
const int N=20010,M=50010;
int n,m,q,cnt,i,x,y;
map<int,int>T[20010];
int sgn(double x){
  if(fabs(x)<eps)return 0;
  return x>0?1:-1;
}
struct P{
  double x,y;
  P(){}
  P(double _x,double _y){x=_x,y=_y;}
  double operator*(const P&b){return x*b.y-y*b.x;}
}a[N],b[N];
struct E{
  int x,y;double o;
  E(){}
  E(int _x,int _y){x=_x,y=_y,o=atan2(a[y].x-a[x].x,a[y].y-a[x].y);}
}e[M];
bool del[M],ex[M];int from[M],id[N];
struct EV{
  double x;int y,t;
  EV(){}
  EV(double _x,int _y,int _t){x=_x,y=_y,t=_t;}
}ev[M<<1];
bool cmpEV(const EV&a,const EV&b){
  if(sgn(a.x-b.x))return a.x<b.x;
  return a.t<b.t;
}
namespace GetArea{
struct cmp{bool operator()(int a,int b){return e[a].o<e[b].o;}};
set<int,cmp>g[N];set<int,cmp>::iterator k;int i,j,q[M],t;
void work(){
  for(i=0;i<m+m;i++)if(!del[i]&&!ex[i]){
    for(q[t=1]=j=i;;q[++t]=j=*k){
      k=g[e[j].y].find(j^1);k++;
      if(k==g[e[j].y].end())k=g[e[j].y].begin();
      if(*k==i)break;
    }
    double s=0;
    for(j=1;j<=t;j++)s+=a[e[q[j]].x]*a[e[q[j]].y],del[q[j]]=1;
    if(sgn(s)<0)continue;
    for(cnt++,j=1;j<=t;j++)from[q[j]]=cnt;
  }
}
}
namespace ScanLine{
struct cmp{
  bool operator()(int A,int B){
    if(e[A].x==e[B].x)return e[A].o>e[B].o;
    double x=min(a[e[A].x].x,a[e[B].x].x),
           yA=(a[e[A].x].y-a[e[A].y].y)*(x-a[e[A].y].x)/
```

```
            (a[e[A].x].x-a[e[A].y].x)+a[e[A].y].y,
          yB=(a[e[B].x].y-a[e[B].y].y)*(x-a[e[B].y].x)/
            (a[e[B].x].x-a[e[B].y].x)+a[e[B].y].y;
    return yA>yB;
  }
};
set<int,cmp>T;
int cnt,i,j,k,g[M],v[M],nxt[M],ed,vis[N],t,tmp[N];
bool cmpC(int x,int y){return a[x].x<a[y].x;}
void add(int x,int y){v[++ed]=y;nxt[ed]=g[x];g[x]=ed;}
void dfs(int x){
  vis[x]=1;
  if(a[x].y>a[t].y)t=x;
  for(int i=g[x];i;i=nxt[i])if(!vis[v[i]])dfs(v[i]);
}
double cal(int A,double x){
  return(a[e[A].x].y-a[e[A].y].y)*(x-a[e[A].y].x)/
        (a[e[A].x].x-a[e[A].y].x)+a[e[A].y].y;
}
void connect(){
  for(i=0;i<m+m;i++)add(e[i].x,e[i].y);
  for(i=1;i<=n;i++)if(!vis[i])dfs(t=i),ev[cnt++]=EV(a[t].x,t,2);
  for(i=0;i<m+m;i++)if(sgn(a[e[i].x].x-a[e[i].y].x)>0){
    ev[cnt++]=EV(a[e[i].y].x,i,1);
    ev[cnt++]=EV(a[e[i].x].x,i,0);
  }
  sort(ev,ev+cnt,cmpEV);
  a[n+1]=P(10010,10010);
  a[n+2]=P(-10010,10010);
  e[m+m]=E(n+1,n+2);
  T.insert(m+m);
  e[m+m+1]=E(n+2,n+1);
  n+=2,m++;
  for(ed=0,i=1;i<=n;i++)g[i]=0;
  for(i=0;i<cnt;i++){
    if(ev[i].t==0)T.erase(ev[i].y);
    if(ev[i].t==1)T.insert(ev[i].y);
    if(ev[i].t==2){
      a[n+1]=P(ev[i].x,a[ev[i].y].y+eps);
      a[n+2]=P(ev[i].x-1,a[ev[i].y].y+eps);
      e[m+m]=E(n+1,n+2);
      T.insert(m+m);
      set<int,cmp>::iterator j=T.find(m+m);
      j--,add(*j,ev[i].y);
      T.erase(m+m);
    }
  }
  int newm=m+m;
  for(i=0;i<m+m;i++){
    for(cnt=0,j=g[i];j;j=nxt[j]){
      if(!sgn(a[v[j]].x-a[e[i].x].x)){
        e[newm++]=E(v[j],e[i].x);
        e[newm++]=E(e[i].x,v[j]);
        continue;
      }
      if(!sgn(a[v[j]].x-a[e[i].y].x)){
        e[newm++]=E(v[j],e[i].y);
```

```
        e[newm++]=E(e[i].y,v[j]);
        continue;
      }
      tmp[++cnt]=v[j];
    }
    if(!cnt)continue;
    ex[i]=ex[i^1]=1;
    sort(tmp+1,tmp+cnt+1,cmpC);
    for(k=e[i].y,j=1;j<=cnt;k=n,j++){
      a[++n]=P(a[tmp[j]].x,cal(i,a[tmp[j]].x));
      e[newm++]=E(k,n);
      e[newm++]=E(n,k);
      e[newm++]=E(tmp[j],n);
      e[newm++]=E(n,tmp[j]);
    }
    e[newm++]=E(n,e[i].x);
    e[newm++]=E(e[i].x,n);
  }
  m=newm/2;
}
void location(){
  for(i=cnt=0;i<m+m;i++)if(!ex[i]&&sgn(a[e[i].x].x-a[e[i].y].x)>0){
    ev[cnt++]=EV(a[e[i].y].x,i,1);
    ev[cnt++]=EV(a[e[i].x].x,i,0);
  }
  for(i=0;i<q;i++)ev[cnt++]=EV(b[i].x,i,2);
  sort(ev,ev+cnt,cmpEV);
  T.clear();
  for(i=0;i<cnt;i++){
    if(ev[i].t==0)T.erase(ev[i].y);
    if(ev[i].t==1)T.insert(ev[i].y);
    if(ev[i].t==2){
      a[n+1]=P(ev[i].x,b[ev[i].y].y);
      a[n+2]=P(ev[i].x-1,b[ev[i].y].y);
      e[m+m]=E(n+1,n+2);
      T.insert(m+m);
      set<int,cmp>::iterator j=T.find(m+m);
      if(j!=T.begin())j--,id[ev[i].y]=from[*j];
      T.erase(m+m);
    }
  }
}
}
int getid(){
  int x,y;
  scanf("%d%d",&x,&y);
  if(T[x+10000][y])return T[x+10000][y];
  T[x+10000][y]=++n;
  a[n]=P(x,y);
  return n;
}
int main(){
  scanf("%d%d",&q,&m);
  for(i=0;i<q;i++)scanf("%lf%lf",&b[i].x,&b[i].y);
  for(i=0;i<m;i++){
    x=getid();
    y=getid();
```

```
    e[i<<1]=E(x,y);
    e[i<<1|1]=E(y,x);
  }
  ScanLine::connect();//通过加辅助线使得图连通
  for(i=0;i<m+m;i++)if(!ex[i])GetArea::g[e[i].x].insert(i);
  GetArea::work();//求出所有域
  ScanLine::location();//利用扫描线进行点定位
}
```

11 黑科技与杂项

11.1 开栈

11.1.1 32 位 Win 下

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
extern int main2(void) __asm__ ("_main2");
int main2(){
  char test[255<<20];
  memset(test,42,sizeof(test));
  printf(":)\n");
  exit(0);//注意：这里需要exit(0);来退出程序，否则会得到非零退出的错误，可能报RE的
}
int main(){
  int size=256<<20;//256MB
  char*p=(char*)malloc(size)+size;
  __asm__ __volatile__(
    "movl  %0, %%esp\n"
    "pushl $_exit\n"
    "jmp _main2\n"
    :: "r"(p));
}
```

11.1.2 64 位 Linux 下：(对 main() 中的汇编语句做修改)

```
  __asm__ __volatile__(
    "movq  %0, %%rsp\n"
    "pushq $exit\n"
    "jmp main2\n"  //注意到下需要，下要用win_main2linuxmain2
    :: "r"(p));
```

11.1.3 简化版本

一定要最后写一句 exit(0); 退出程序。

```
int size=256<<20;//256MB
char*p=(char*)malloc(size)+size;
__asm__ __volatile__("movq  %0, %%rsp\n" :: "r"(p));//64bit
```

11.2 I/O 优化

11.2.1 普通 I/O 优化

```
//适用于非负整数
template<class T>
void scan_d(T&ret){
  char c;ret=0;
```

```
  while((c=getchar())<'0'||c>'9');
  while(c>='0'&&c<='9')ret=ret*10+(c-'0'),c=getchar();
}
//适用于整数
template<class T>
bool scan_d(T&ret){
  char c;int sgn;
  if(c=getchar(),c==EOF)return 0;//EOF
  while(c!='-'&&(c<'0'||c>'9'))c=getchar();
  sgn=(c=='-')?-1:1;
  ret=(c=='-')?0:(c-'0');
  while(c=getchar(),c>='0'&&c<='9')ret=ret*10+(c-'0');
  ret*=sgn;
  return 1;
}
//适用于整数,(int,long long,float,double)
template<class T>
bool scan_d(T&ret){
  char c;int sgn;T bit=0.1;
  if(c=getchar(),c==EOF)return 0;
  while(c!='-'&&c!='.'&&(c<'0'||c>'9'))c=getchar();
  sgn=(c=='-')?-1:1;
  ret=(c=='-')?0:(c-'0');
  while(c=getchar(),c>='0'&&c<='9')ret=ret*10+(c-'0');
  if(c==' '||c=='\n'){ret*=sgn;return 1;}
  while(c=getchar(),c>='0'&&c<='9')ret+=(c-'0')*bit,bit/=10;
  ret*=sgn;
  return 1;
}
//输出外挂
void out(int x){
  if(x>9)out(x/10);
  putchar(x%10+'0');
}
```

11.2.2 文艺 I/O 优化

```
const int BUFSIZE=20<<20;//<<10KB,<<20MB
char Buf[BUFSIZE+1],*buf=Buf;
//fread(Buf,1,BUFSIZE,stdin)在读入之前写这句话;
//非负整数
template<class T>
void scan(T&a){
  for(a=0;*buf<'0'||*buf>'9';buf++);
  while(*buf>='0'&&*buf<='9'){a=a*10+(*buf-'0');buf++;}
}
//任何整数
template<class T>
void scan(T&a){
  int sgn=1;
  for(a=0;*buf<'0'||*buf>'9';buf++)if(*buf=='-')sgn=-1;
  while(*buf>='0'&&*buf<='9'){a=a*10+(*buf-'0');buf++;}
  a*=sgn;
}
//支持EOF，非负整数
```

```
const int BUFSIZE=20<<20;//<<10KB,<<20MB
char Buf[BUFSIZE+1],*buf=Buf;
size_t lastlen=0;
template<class T>
bool scan(T&a){
  for(a=0;(*buf<'0');buf++);
  if(buf-Buf>=lastlen)return 0;
  while(*buf>='0'){a=a*10+(*buf-'0');buf++;}
  return 1;
}
//lastlen=fread(Buf,1,BUFSIZE,stdin);在读入之前写这句话
//读入字符串
void scanString(char*c){
  for(;*buf=='\n'||*buf=='\t'||*buf==' ';buf++);
  while((*buf!='\n')&&(*buf!='\t')&&(*buf!=' ')){*(c++)=*(buf++);}
  *c=0;
}
```

11.2.3 二逼 I/O 优化

```
namespace IO{
const int MX=4096;
char buf[MX],t[50];
int bi=MX,bn=MX;
int read(char*s){//读入字符串
  while(bn){
    for(;bi<bn&&buf[bi]<=' ';bi++);
    if(bi<bn)break;
    bn=fread(buf,1,MX,stdin);
    bi=0;
  }
  int sn=0;
  while(bn){
    for(;bi<bn&&buf[bi]>' ';bi++)s[sn++]=buf[bi];
    if(bi<bn)break;
    bn=fread(buf,1,MX,stdin);
    bi=0;
  }
  s[sn]=0;
  return sn;
}
bool read(int&x){//读入并转化成变量，这里以int为例
  if(!read(t))return 0;
  x=atoi(t);//long long和double的读入同理，有atoll()和atof()
  return 1;
}
}
```

11.3 位运算及其运用

11.3.1 枚举子集

枚举 i 的非空子集 j。

```
for(j=i;j;j=(j-1)&i);
```

11.3.2 求 1 的个数

```
int __builtin_popcount(unsigned int x);
```

11.3.3 求前缀 0 的个数

```
int __builtin_clz(unsigned int x);
```

11.3.4 求后缀 0 的个数

```
int __builtin_ctz(unsigned int x);
```

11.4 石子合并

每次可以合并相邻两堆石子，代价为两堆石子的个数和，用 GarsiaWachs 算法求最小总代价。

```
#include<cstdio>
int a[50003],n,i,t,ans;
void combine(int k){
  int tmp=a[k]+a[k-1],i,j;ans+=tmp;
  for(i=k;i<t-1;i++)a[i]=a[i+1];
  for(t--,j=k-1;j>0&&a[j-1]<tmp;j--)a[j]=a[j-1];
  a[j]=tmp;
  while(j>=2&&a[j]>=a[j-2])i=t-j,combine(j-1),j=t-i;
}
int main(){
  while(1){
    scanf("%d",&n);
    if(!n)return 0;
    for(i=ans=0;i<n;i++)scanf("%d",a+i);
    for(t=i=1;i<n;i++){
      a[t++]=a[i];
      while(t>=3&&a[t-3]<=a[t-1])combine(t-2);
    }
    while(t>1)combine(t-1);
    printf("%d\n",ans);
  }
}
```

11.5 最小乘积生成树

把方案看成一个二维点，$x = \sum a, y = \sum b$，答案一定在下凸壳上。

找到 l, r 两个点，l 是 x 最小的，r 是 y 最小的，然后递归调用 $work(l, r)$：

1. 找到离该直线最远的点，那个点一定在下凸壳上。

2. 将边权设为 (a, b) 叉积 $(l - r)$。

3. 求出最小生成树作为点 mid。

4. 递归 $work(l, mid)$ 和 $work(mid, r)$。

```
#include<cstdio>
#include<algorithm>
#define N 210
#define M 10010
using namespace std;
typedef long long ll;
struct P{
  int x,y;P(){x=y=0;}P(int _x,int _y){x=_x,y=_y;}
  P operator-(const P&a){return P(x-a.x,y-a.y);}
}l,r;
ll cross(P a,P b){return (ll)a.x*(ll)b.y-(ll)a.y*(ll)b.x;}
struct E{int x,y,a,b,c;}a[M];
bool cmp(const E&a,const E&b){return a.c<b.c;}
int n,m,i,f[N];
ll ans=1000000000000000LL;
int F(int x){return f[x]==x?x:f[x]=F(f[x]);}
P kruskal(){
  P p;int i;
  sort(a+1,a+m+1,cmp);
  for(i=1;i<=n;i++)f[i]=i;
  for(i=1;i<=m;i++)if(F(a[i].x)!=F(a[i].y))
    f[f[a[i].x]]=f[a[i].y],p.x+=a[i].a,p.y+=a[i].b;
  if((ll)p.x*(ll)p.y<ans)ans=(ll)p.x*(ll)p.y;
  return p;
}
void work(P l,P r){
  P t=l-r;
  for(int i=1;i<=m;i++)a[i].c=cross(P(a[i].a,a[i].b),t);
  P mid=kruskal();
  if(cross(mid-l,r-mid)>0)work(l,mid),work(mid,r);
}
int main(){
  scanf("%d%d",&n,&m);
  for(i=1;i<=m;i++)scanf("%d%d%d%d",&a[i].x,&a[i].y,&a[i].a,&a[i].b),a[i].x++,a[i].y++;
  for(i=1;i<=m;i++)a[i].c=a[i].a;
  l=kruskal();
  for(i=1;i<=m;i++)a[i].c=a[i].b;
  r=kruskal();
  work(l,r);
  printf("%lld",ans);
}
```

11.6 特征多项式加速线性递推

$a[n]=\sum_{i=1}^{m}c[m-i]a[n-i]$，给定 $a[0],a[1],...,a[m-1]$，求 $a[n]$。

用特征多项式 + 快速幂加速线性递推，时间复杂度 $O(m^2\log n)$。

```
const int M=2000,P=1000000007;
int n,m,i,j,x,w,b,t,a[M],c[M],v[M],u[M<<1],ans;
int main(){
  scanf("%d%d",&n,&m);
  for(i=m-1;~i;i--)scanf("%d",&c[i]),c[i]=(c[i]%P+P)%P;
  for(i=0;i<m;i++)scanf("%d",&a[i]),a[i]=(a[i]%P+P)%P;
  for(i=0;i<m;i++)v[i]=1;
  for(w=!!n,i=n;i>1;i>>=1)w<<=1;
  for(x=0;w;copy(u,u+m,v),w>>=1,x<<=1){
    fill_n(u,m<<1,0),b=!!(n&w),x|=b;
    if(x<m)u[x]=1;
    else{
      for(i=0;i<m;i++)for(j=0,t=i+b;j<m;j++,t++)u[t]=((ll)v[i]*v[j]+u[t])%P;
      for(i=(m<<1)-1;i>=m;i--)for(j=0,t=i-m;j<m;j++,t++)u[t]=((ll)c[j]*u[i]+u[t])%P;
    }
  }
  for(i=0;i<m;i++)ans=((ll)v[i]*a[i]+ans)%P;
  printf("%d",ans);
}
```

11.7 三元环的枚举

给定一张 n 个点 m 条边的无向图，在 $O(m\sqrt{m})$ 的时间内枚举所有三元环。

```
const int N=100010,M=200010,Base=(1<<21)-1;
struct edge{int v,w;edge*nxt;}epool[M],*ecur=epool,*g[N],*j,*k;
struct Edge{int x,y,w;Edge*nxt;}Epool[M],*Ecur=Epool,*G[Base+1],*l;
int n,m,i,d[N],x,y,lim,Hash;
struct Elist{int x,y,w;}e[M];
bool cmp(const Elist&a,const Elist&b){return a.x==b.x?a.y<b.y:a.x<b.x;}
int vis(int x,int y){
  for(l=G[(x<<8|y)&Base];l;l=l->nxt)if(l->x==x&&l->y==y)return l->w;
  return 0;
}
int main(){
  while(~scanf("%d%d",&n,&m)){
    while(lim*lim<m)lim++;
    for(i=1;i<=m;i++){
      scanf("%d%d",&x,&y);
      if(x<y)swap(x,y);
      e[i].x=x,e[i].y=y;
    }
    for(sort(e+1,e+m+1,cmp),i=1;i<=m;i++){
      d[x=e[i].x]++;
      ecur->v=y=e[i].y;ecur->w=i;ecur->nxt=g[x];g[x]=ecur++;
      Ecur->x=x;Ecur->y=y;Ecur->w=i;Ecur->nxt=G[Hash=(x<<8|y)&Base];G[Hash]=Ecur++;
    }
    for(i=3;i<=n;i++)for(j=g[i];j;j=j->nxt)if(d[x=j->v]<=lim){
      for(k=g[x];k;k=k->nxt)if(y=vis(i,k->v)){
        //三条边分别为e[j->w] e[k->w] e[y]
```

```
        //与x点相连的两条边分别为e[j->w] e[k->w]
        //与i点相连的两条边分别为e[j->w] e[y]
        //与k->v点相连的两条分别边为e[k->w] e[y]
      }
    }else for(k=j->nxt;k;k=k->nxt)if(y=vis(x,k->v)){
        //三条边分别为e[j->w] e[k->w] e[y]
        //与i点相连的两条边分别为e[j->w] e[k->w]
        //与x点相连的两条边分别为e[j->w] e[y]
        //与k->v点相连的两条边分别为e[k->w] e[y]
    }
    lim=0,ecur=epool,Ecur=Epool;
    for(i=1;i<=n;i++)d[i]=0,g[i]=NULL;
    for(i=1;i<=m;i++)G[(e[i].x<<8|e[i].y)&Base]=NULL;
  }
}
```

11.8 所有区间 gcd 的预处理

```
int n,i,j,a[N],l[N],v[N];
int fun(int x,int y){return __gcd(x,y);}
int main(){
  for(scanf("%d",&n),i=1;i<=n;i++)scanf("%d",&a[i]);
  for(i=1;i<=n;i++)for(v[i]=a[i],j=l[i]=i;j;j=l[j]-1){
    v[j]=fun(v[j],a[i]);
    while(l[j]>1&&fun(a[i],v[l[j]-1])==fun(a[i],v[j]))l[j]=l[l[j]-1];
    //[l[j]..j,i]区间内的值求fun均为v[j]
  }
}
```

11.9 无向图最小割

给定一张 n 个点 m 条带权边的无向图，点的编号为 0 到 $n-1$，用 Stoer-Wagner 算法求它的最小割，即最后的图至少有 2 个连通块。

```
const int N=502,inf=1000000000;
int v[N],w[N],c[N],g[N][N],S,T,now,n,m,x,y,z;
void search(){
  int i,j,k,t;
  for(i=0;i<n;i++)v[i]=w[i]=0;
  for(S=T=-1,i=0;i<n;i++){
    for(k=-inf,j=0;j<n;j++)if(!c[j]&&!v[j]&&w[j]>k)k=w[t=j];
    if(T==t)return;
    S=T,T=t,now=k,v[t]=1;
    for(j=0;j<n;j++)if(!c[j]&&!v[j])w[j]+=g[t][j];
  }
}
int stoerwagner(){
  int i,j,ans=inf;
  for(i=0;i<n;i++)c[i]=0;
  for(i=0;i<n-1;i++){
    search();
    if(now<ans)ans=now;
    if(ans==0)return 0;
```

```
    for(c[T]=1,j=0;j<n;j++)if(!c[j])g[S][j]+=g[T][j],g[j][S]+=g[j][T];
  }
  return ans;
}
int main(){
  scanf("%d%d",&n,&m);
  while(m--)scanf("%d%d%d",&x,&y,&z),g[x][y]+=z,g[y][x]+=z;
  printf("%d",stoerwagner());
}
```

11.10 分割回文串

输入一个长度为 n，从 0 开始的字符串 s，MinPalindromeSpilt(s) 后，$f[n][0]$ 表示该串分成偶数个回文串，至少能分成几个；$f[n][1]$ 表示该串分成奇数个回文串，至少能分成几个。若能分成 X 个，那么一定可以分成 $X+2$ 个。

```
char s[N];
int d[N][2],f[N][2];
struct P{
  int d[3];
  P(){}
  P(int a,int b,int c){d[0]=a;d[1]=b;d[2]=c;}
  int&operator[](int x){return d[x];}
}a[32],b[32],c[32];
void up(int f[][2],int x,int y){
  if(y<=0)return;
  int p=y&1;
  if(f[x][p]<0)f[x][p]=y;else f[x][p]=min(f[x][p],y);
}
void make(int f[][2],int x,int y){if(y>0)f[x][y&1]=y;}
void MinPalindromeSpilt(char*s){
  int n=strlen(s);
  memset(a,0,sizeof a);
  memset(b,0,sizeof b);
  memset(c,0,sizeof c);
  memset(d,0,sizeof d);
  memset(f,0,sizeof f);
  for(int i=0;i<=n;i++)d[i][0]=1000000000,d[i][1]=1000000001;
  for(int ca=0,j=0;j<n;j++){
    int cb=0,cc=0,r=-j-2;
    for(int u=0;u<ca;u++){
      int i=a[u][0];
      if(i>=1&&s[i-1]==s[j])a[u][0]--,b[cb++]=a[u];
    }
    for(int u=0;u<cb;u++){
      int i=b[u][0],d=b[u][1],k=b[u][2];
      if(i-r!=d){
        c[cc++]=P(i,i-r,1);
        if(k>1)c[cc++]=P(i+d,d,k-1);
      }else c[cc++]=P(i,d,k);
      r=i+(k-1)*d;
    }
    if(j>=1&&s[j-1]==s[j])c[cc++]=P(j-1,j-1-r,1),r=j-1;
    c[cc++]=P(j,j-r,1),ca=0;
```

```
    P&h=c[0];
    for(int u=1;u<cc;u++){
      P&x=c[u];
      if(x[1]==h[1])h[2]+=x[2];else a[ca++]=h,h=x;
    }
    a[ca++]=h;
    if((j+1)%2==0)f[j+1][0]=j+1,f[j+1][1]=1000000001;
    else f[j+1][0]=1000000000,f[j+1][1]=j+1;
    for(int u=0;u<ca;u++){
      int i=a[u][0],e=a[u][1],k=a[u][2];
      r=i+(k-1)*e;
      up(f,j+1,f[r][0]+1),up(f,j+1,f[r][1]+1);
      if(k>1)up(f,j+1,d[i+1-e][0]),up(f,j+1,d[i+1-e][1]);
      if(i+1-e>=0){
        if(k>1)up(d,i+1-e,f[r][0]+1),up(d,i+1-e,f[r][1]+1);
        else make(d,i+1-e,f[r][0]+1),make(d,i+1-e,f[r][1]+1);
      }
    }
  }
}
int main(){
  int T;
  for(scanf("%d",&T);T--;){
    scanf("%s",s);
    MinPalindromeSpilt(s);
    int n=strlen(s);
    printf("%d %d\n",f[n][0],f[n][1]);
  }
}
```

11.11 高精度计算

```
#include<algorithm>
using namespace std;
const int N_huge=850,base=100000000;
char s[N_huge*10];
struct huge{
    typedef long long value;
    value a[N_huge];int len;
    void clear(){len=1;a[len]=0;}
    huge(){clear();}
    huge(value x){*this=x;}
    huge operator =(huge b){
        len=b.len;for (int i=1;i<=len;++i)a[i]=b.a[i]; return *this;
    }
    huge operator +(huge b){
        int L=len>b.len?len:b.len;huge tmp;
        for (int i=1;i<=L+1;++i)tmp.a[i]=0;
        for (int i=1;i<=L;++i){
            if (i>len)tmp.a[i]+=b.a[i];
            else if (i>b.len)tmp.a[i]+=a[i];
            else {
                tmp.a[i]+=a[i]+b.a[i];
                if (tmp.a[i]>=base){
                    tmp.a[i]-=base;++tmp.a[i+1];
```

```
                }
            }
        }
        if (tmp.a[L+1])tmp.len=L+1;
            else tmp.len=L;
        return tmp;
    }
    huge operator -(huge b){
        int L=len>b.len?len:b.len;huge tmp;
        for (int i=1;i<=L+1;++i)tmp.a[i]=0;
        for (int i=1;i<=L;++i){
            if (i>b.len)b.a[i]=0;
            tmp.a[i]+=a[i]-b.a[i];
            if (tmp.a[i]<0){
                tmp.a[i]+=base;--tmp.a[i+1];
            }
        }
        while (L>1&&!tmp.a[L])--L;
        tmp.len=L;
        return tmp;
    }
    huge operator *(huge b){
        int L=len+b.len;huge tmp;
        for (int i=1;i<=L;++i)tmp.a[i]=0;
        for (int i=1;i<=len;++i)
            for (int j=1;j<=b.len;++j){
                tmp.a[i+j-1]+=a[i]*b.a[j];
                if (tmp.a[i+j-1]>=base){
                    tmp.a[i+j]+=tmp.a[i+j-1]/base;
                    tmp.a[i+j-1]%=base;
                }
            }
        tmp.len=len+b.len;
        while (tmp.len>1&&!tmp.a[tmp.len])--tmp.len;
        return tmp;
    }
    pair<huge,huge> divide(huge a,huge b){
        int L=a.len;huge c,d;
        for (int i=L;i;--i){
            c.a[i]=0;d=d*base;d.a[1]=a.a[i];
            //while (d>=b){d-=b;++c.a[i];}
            int l=0,r=base-1,mid;
            while (l<r){
                mid=(l+r+1)>>1;
                if (b*mid<=d)l=mid;
                    else r=mid-1;
            }
            c.a[i]=l;d-=b*l;
        }
        while (L>1&&!c.a[L])--L;c.len=L;
        return make_pair(c,d);
    }
    huge operator /(value x){
        value d=0;huge tmp;
        for (int i=len;i;--i){
            d=d*base+a[i];
            tmp.a[i]=d/x;d%=x;
```

```
        }
        tmp.len=len;
        while (tmp.len>1&&!tmp.a[tmp.len])--tmp.len;
        return tmp;
    }
    value operator %(value x){
        value d=0;
        for (int i=len;i;--i)d=(d*base+a[i])%x;
        return d;
    }
    huge operator /(huge b){return divide(*this,b).first;}
    huge operator %(huge b){return divide(*this,b).second;}
    huge &operator +=(huge b){*this=*this+b;return *this;}
    huge &operator -=(huge b){*this=*this-b;return *this;}
    huge &operator *=(huge b){*this=*this*b;return *this;}
    huge &operator ++(){huge T;T=1;*this=*this+T;return *this;}
    huge &operator --(){huge T;T=1;*this=*this-T;return *this;}
    huge operator ++(int){huge T,tmp=*this;T=1;*this=*this+T;return tmp;}
    huge operator --(int){huge T,tmp=*this;T=1;*this=*this-T;return tmp;}
    huge operator +(value x){huge T;T=x;return *this+T;}
    huge operator -(value x){huge T;T=x;return *this-T;}
    huge operator *(value x){huge T;T=x;return *this*T;}
    //huge operator /(value x){huge T;T=x;return *this/T;}
    //huge operator %(value x){huge T;T=x;return *this%T;}
    huge operator *=(value x){*this=*this*x;return *this;}
    huge operator +=(value x){*this=*this+x;return *this;}
    huge operator -=(value x){*this=*this-x;return *this;}
    huge operator /=(value x){*this=*this/x;return *this;}
    huge operator %=(value x){*this=*this%x;return *this;}
    bool operator ==(value x){huge T;T=x;return *this==T;}
    bool operator !=(value x){huge T;T=x;return *this!=T;}
    bool operator <=(value x){huge T;T=x;return *this<=T;}
    bool operator >=(value x){huge T;T=x;return *this>=T;}
    bool operator <(value x){huge T;T=x;return *this<T;}
    bool operator >(value x){huge T;T=x;return *this>T;}
    huge operator =(value x){
        len=0;
        while (x)a[++len]=x%base,x/=base;
        if (!len)a[++len]=0;
        return *this;
    }
    bool operator <(huge b){
        if (len<b.len)return 1;
        if (len>b.len)return 0;
        for (int i=len;i;--i){
            if (a[i]<b.a[i])return 1;
            if (a[i]>b.a[i])return 0;
        }
        return 0;
    }
    bool operator ==(huge b){
        if (len!=b.len)return 0;
        for (int i=len;i;--i)
            if (a[i]!=b.a[i])return 0;
        return 1;
    }
    bool operator !=(huge b){return !(*this==b);}
```

```
    bool operator >(huge b){return !(*this<b||*this==b);}
    bool operator <=(huge b){return (*this<b)||(*this==b);}
    bool operator >=(huge b){return (*this>b)||(*this==b);}
    void str(char s[]){
        int l=strlen(s);value x=0,y=1;len=0;
        for (int i=l-1;i>=0;--i){
            x=x+(s[i]-'0')*y;y*=10;
            if (y==base)a[++len]=x,x=0,y=1;
        }
        if (!len||x)a[++len]=x;
    }
    void read(){
        scanf("%s",s);this->str(s);
    }
    void print(){
        printf("%d",(int)a[len]);
        for (int i=len-1;i;--i){
            for (int j=base/10;j>=10;j/=10){
                if (a[i]<j)printf("0");
                    else break;
            }
            printf("%d",(int)a[i]);
        }
        printf("\n");
    }
}f[1005];
int main(){
  f[1]=f[2]=1;
  for(int i=3;i<=1000;i++)f[i]=f[i-1]+f[i-2];
}
```

11.12 Rope

push_back(x)：在末尾添加 x(x 是 char)

insert(pos,x)：在 pos 插入 x (x 是字符串,x 后面加个 int 参数可以只能 x 中插入几个)

erase(pos,x)：从 pos 开始删除 x 个

replace(pos,x)：从 pos 开始换成 x(x 是字符串,x 后面加个 int 参数可以只能 x 中的前几个)

substr(pos,x)：提取 pos 开始 x 个

copy(x)：复制 rope 中所有内容到 x 字符串

at(x)/[x]：访问第 x 个元素

注意事项：

1、rope 好像并不原声支持对一个字符串复制 n 遍的做法，要自己手写快速幂

2、rope 可以用 += 来做追加操作

3、rope 中访问、修改一个特定字符的操作是 $O(\log length)$ 的

4、rope 中 [] 运算符只能访问不能修改，需要修改要用 mutable_begin()+ 偏移量，得到迭代器再修改

11.12.1 示例 1

展开一个括号压缩的字符串。比如 z(rz)3r(rui)2cumt 展开为 zrzrzrzrruiruicumt。

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
#include<limits.h>
#include<math.h>
#include<algorithm>
using namespace std;
typedef long long ll;
#include<ext/rope> //header with rope
using namespace __gnu_cxx;//namespace with rope and some additional stuff
rope<char> tillRight(char *str,int &p,int &times){
  rope<char> ans=””; times=0;
  while(str[p]!=’)’) ans+=str[p++];
  p++;
  while(isdigit(str[p])){
    times=times*10+(str[p++]-’0’);
  }
  p--;
  return ans;
}
rope<char> times(rope<char> &src,int times){
  rope<char> ans,tmp=src;
  while(times){
    if(times&1) ans+=tmp;
    times>>=1; tmp+=tmp;
  }
  return ans;
}
void expand(char * str, rope<char>& ss){
  int p=0;
  ss.clear();
  for(;str[p];p++){
    if(str[p]!=’(’) ss+=str[p];
    else{
      p++;
      int t;
      crope tmp=tillRight(str,p,t);
      ss.append(times(tmp,t));
    }
  }
}
char str[20005];
int main(){
  while(~scanf(”%s”,str)){
    rope<char> txt;
    expand(str,txt);
    printf(”%s\n”,txt.c_str());
  }
}
```

11.12.2 示例 2

给一个 100000 长的串和 100000 次查询，每次要把 $[l, r]$ 内的元素移动到序列开头。输出最后的序列。

```
#include<iostream>
#include<cstdio>
#include<ext/rope> //header with rope
using namespace std;
using namespace __gnu_cxx;//namespace with rope and some additional stuff
int main(){
  ios_base::sync_with_stdio(false);
  rope <int> v;//use as usual STL container
  int n, m;
  cin >> n >> m;
  for(int i = 1; i <= n; ++i)v.push_back(i);//initialization
  int l, r;
  for(int i = 0; i < m; ++i){
    cin >> l >> r;
    --l, --r;
    rope <int> cur = v.substr(l, r - l + 1);
    v.erase(l, r - l + 1);
    v.insert(v.mutable_begin(), cur);
  }
  for(rope <int>::iterator it = v.mutable_begin(); it != v.mutable_end(); ++it)
    cout << *it << " ";
}
```

11.13 pb_ds 的红黑树

```
#include<algorithm>
using namespace std;
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_cxx;
using namespace __gnu_pbds;
#define rbset(T) tree<T,null_type,less<T>,rb_tree_tag,tree_order_statistics_node_update>
struct RBTree{
  rbset(int) rb;
  void init(){
    rb=rbset(int)();
  }
  void insert(int x){//插入
    rb.insert(x);
  }
  void remove(int x){//删除
    rb.erase(x);
  }
  int findKth(int x){//找第k大(k从0开始)
    return *rb.find_by_order(x);
  }
  int findElementRank(int x){//找>=x的第一个元素是排第几
    return rb.order_of_key(x);
  }
```

```
};
/*
    ordered_set X;
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true

    cout<<X.order_of_key(-5)<<endl;  // 0
    cout<<X.order_of_key(1)<<endl;   // 0
    cout<<X.order_of_key(3)<<endl;   // 2
    cout<<X.order_of_key(4)<<endl;   // 2
    cout<<X.order_of_key(400)<<endl; // 5
*/
```

12 Java

12.1 输入

12.1.1 声明一个输入对象 cin

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

12.1.2 输入一个 int 值

```
Int a=cin.nextInt();
```

12.1.3 输入一个大数

```
BigDecimal a=cin.nextBigDecimal();
```

12.1.4 EOF 结束

```
while(cin.hasNext())…{}
```

12.2 输出

输出任意类型的 str。

```
System.out.println(str);//有换行
System.out.print(str);//无换行
System.out.println”(“str);//输出字符串str
System.out.printf(”Hello,%s.Next year,you’ll be %d”,name,age);//C风格输出(无C风格输入)
```

12.3 大数类

12.3.1 赋值

```
BigInteger a=BigInteger.valueOf(12);
BigInteger b=new BigInteger(String.valueOf(12));
BigDecimal c=BigDecimal.valueOf(12.0);
BigDecimal d=new BigDecimal(”12.0”);//建议使用字符串以防止double类型导致的误差
```

也可以用上述方法构造一个临时对象用于参与运算。

```
b.add(BigInteger.valueOf(105));
```

12.3.2 比较

```
c.compareTo(BigDecimal.ZERO)==0//判断相等，c 等于0
c.compareTo(BigDecimal.ZERO)>0//判断大于，c大于0
c.compareTo(BigDecimal.ZERO)<0//判断小于，c小于0
```

12.3.3 基本运算

```
Big*** add(Big*** b)//加上b
Big*** subtract(Big*** b)//减去b
Big*** multiply(Big*** b)//乘b
Big*** divided(Big*** b)//除b
Big*** pow(int b)//计算this^b，注意b只能是int类型
Big*** remainder(Big*** b)//mod b，即计算this%b
Big*** abs()//返回this的绝对值
Big*** negate()//返回—this
Big*** max(Big*** b)//返回this和b中的最大值
Big*** min(Big*** b)//返回this和b中的最小值
```

BigInteger 特有的函数：

```
gcd(BigInteger val)//返回一个BigInteger，其值是abs(this)和abs(val)的最大公约数
mod(BigInteger val)//求 this mod val
modInverse(BigInteger val)//求逆元，返回this^(—1) mod m
```

12.3.4 BigDecimal 的格式控制

toString() 将 BigDecimal 对象的数值转换成字符串。之后可配合字符串处理函数进行一些处理：

```
str.startWith("0")//以0开始
str.endWith("0")//以0结束
str.subString(int x,int y)//从x到y的str的子串
str.subString(int x))//从x到结尾的str的子串
c.stripTrailingZeros().toPlainString();//c去除末尾0，并转换成普通字符串
```

setScale(int newScale,RoundingMode roundingMode) 返回 BigDecimal，其标度（小数点后保留位数）为指定值，其非标度值通过此 BigDecimal 的非标度值乘以或除以十的适当次幂来确定，以维护其总值。（用法见下例）

CEILING	向正无限大方向舍入的舍入模式
DOWN	向零方向舍入的舍入模式
FLOOR	向负无限大方向舍入的舍入模式
HALF_DOWN	向最接近数字方向舍入的舍入模式，如果与两个相邻数字的距离相等，则向下舍入
HALF_EVEN	向最接近数字方向舍入的舍入模式，如果与两个相邻数字的距离相等，则向相邻的偶数舍入
HALF_UP	向最接近数字方向舍入的舍入模式，如果与两个相邻数字的距离相等，则向上舍入
UNNECESSARY	用于断言请求的操作具有精确结果的舍入模式，因此不需要舍入
UP	远离零方向舍入的舍入模式

12.3.5 创建 BigDecimal 对象

```
BigDecimal bigNumber=new BigDecimal(”89.1234567890123456789”);
BigDecimal bigRate=new BigDecimal(1000);
BigDecimal bigResult=new BigDecimal();//对象bigResult的值为0.0
```

12.3.6 对 bigNumber 的值乘以 1000，结果赋予 bigResult

```
bigResult=bigNumber.multiply(bigRate);
System.out.println(bigResult);
```

12.3.7 BigInteger 的进制转换

Java 支持的进制范围为 2 36(0 9+ 小写的 a z)。

```
BigInteger a=cin.nextBigInteger(2);//读入一个二进制数
System.out.println(a.toString(2));//输出二进制
```

12.4 小数四舍五入

```
import java.util.*;//输入输出所在的包
import java.math.*;//高精度整数/浮点数所在的包
public class Test{
public static void main(String[] args){
   double i=3.856;
   System.out.println(”四舍五入取整:(3.856)=”
     + new BigDecimal(i).setScale(0,BigDecimal.ROUND_HALF_UP));
   System.out.println(”四舍五入保留两位小数:(3.856)=”
     + new BigDecimal(i).setScale(2,BigDecimal.ROUND_HALF_UP));
}
}
```

12.5 高精度小数 A+B，输出最简结果

```
import java.math.*;
import java.util.*;
public class Main{
  public static void main(String []args){
    Scanner cin=new Scanner(System.in);
    BigDecimal a,b,c;
    while(cin.hasNext()){
      a=cin.nextBigDecimal();b=cin.nextBigDecimal();
      c=a.add(b);
      if(c.compareTo(BigDecimal.ZERO)==0){System.out.println("0"); continue;}
      //不能省，因为stripTrailingZeros()不能很好处理0.00这种情况(JDK8才修复)
      String str=c.stripTrailingZeros().toPlainString();
      if(str.endsWith(".")) str=str.substring(0,str.length()-1);
      System.out.println(str);
    }
  }
}
```

12.6 斐波那契数列

```
import java.util.*;
import java.math.*;
public class Main
{
    public static void main(String[] args){
        BigInteger f[] = new BigInteger[1005];//数组的用法和C#类似
        //二维数组BigInteger[][] f=new BigInteger[1005][1005];
        f[1]=BigInteger.valueOf(1);f[2]=BigInteger.valueOf(1);
        for(int i=3;i<=1000;i++)f[i]=f[i-2].add(f[i-1]);
        Scanner input=new Scanner(System.in);
        int n,t;
        //用for(...;t--;)替代会报错，因为要求第二个表达式必须返回bool
        for(t=input.nextInt();t>0;t--){
            n=input.nextInt();
            System.out.println(f[n]);
        }
    }
}
```

12.7 两个高精度浮点数比较是否相等

```
import java.math.*;
import java.util.*;
public class Main{
    public static void main(String[] args){
        Scanner input=new Scanner(System.in);
        BigDecimal a,b;int result;
        while(input.hasNextBigDecimal()){
            a=input.nextBigDecimal();
            b=input.nextBigDecimal();
```

```
            result=a.compareTo(b);
            System.out.printf(result==0?"YES\r\n":"NO\r\n");
            //方法1：手工处理。win的oj上换行必须\r\n，否则PE
            //Linux下评测时用\n来表换行
            //方法2：Java中%n表示运行平台决定的换行符，智能的输出\r\n或者\n
        }
    }
}
```

12.8 高效的输入类

```
class FastScanner {
    BufferedReader br;
    StringTokenizer st;
    public FastScanner(InputStream in) {
        br = new BufferedReader(new InputStreamReader(in),16384);
        eat("");
    }
    private void eat(String s) {st = new StringTokenizer(s);}
    public String nextLine() {
        try {
            return br.readLine();
        } catch (IOException e) {
            return null;
        }
    }
    public boolean hasNext() {
        while (!st.hasMoreTokens()) {
            String s = nextLine();
            if(s == null)return false;
            eat(s);
        }
        return true;
    }
    public String next() {
        hasNext();
        return st.nextToken();
    }
    public int nextInt() {return Integer.parseInt(next());}
    public double nextDouble() {return Double.parseDouble(next());}
    //需要的其他类型比如long可以仿照
    //BigInteger建议这样写：BigInteger test=new BigInteger(in.next());
    //使用方法：FastScanner in=new FastScanner(System.in);
}
```

12.9 输出外挂

注意输出量很大的时候才有效果，否则会起一定的拖慢反作用。

```
PrintWriter out = new PrintWriter(
                  new BufferedWriter(new OutputStreamWriter(System.out)));
out.println();out.print();//输出使用照常
out.flush();//注意最后一定要追加，不然会WA
```