输入 & 输出

特殊格式

```
long double %Lf
unsigned int %u
unsigned long long %llu

cout << fixed << setprecision(15);
```

文件和流同步

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

ios::sync_with_stdio(false);
cin.tie(0);
```

程序计时

```
fprintf(stderr, "%f\n", (double)clock() / CLOCKS_PER_SEC);
```

整行读入

```
scanf("%[^\n]", s)  // 需测试是否可用
getline(cin, s)
```

读到文件尾

```
while (cin) {}
while (~scanf) {}
```

int128

```
// 需测试是否可用
inline __int128 get128() {
    __int128 x = 0, sgn = 1;
    char c;
    for (c = getchar(); c < '0' || c > '9'; c = getchar()) if (c == '-') sgn = -1;
    for (; c >= '0' && c <= '9'; c = getchar()) x = x * 10 + c - '0';
    return sgn * x;
}

inline void print128(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x >= 10) print128(x / 10);
    putchar(x % 10 + '0');
}
```

读入挂

```
// 本机测试需要EOF才能看到输出结果
#define BUF_SIZE 1048576
```

```
inline char nc() {
    static char buf[BUF_SIZE], *p1 = buf, *p2 = buf;
    if (p1 == p2) {
        p1 = buf;
        p2 = buf + fread(buf, 1, BUF_SIZE, stdin);
        assert(p1 != p2);
    }
    return *p1++;
}

inline bool blank(char c) { return c == ' ' || c == '\n' || c == '\r' || c == '\t'; }

// non-negative integer
inline int getint() {
    int x = 0;
    char c = nc();
    while (blank(c)) c = nc();
    for (; c >= '0' && c <= '9'; c = nc()) x = x * 10 + c - '0';
    return x;
}

// integer
inline int getint() {
    int x = 0, sgn = 1;
    char c = nc();
    while (blank(c)) c = nc();
    if (c == '-') sgn = -1, c = nc();
    for (; c >= '0' && c <= '9'; c = nc()) x = x * 10 + c - '0';
    return sgn * x;
}

#undef BUF_SIZE
```

数据结构

并查集

```
int find(int x) { return (x == pa[x]) ? x : pa[x] = find(pa[x]); }
void merge(int a, int b) { pa[find(a)] = find(b); }
```

RMQ

```
// 下标从0开始
// 一维
struct RMQ {
    int st[MAXN][22]; // 22 = ((int)log2(MAXN) + 1)

    int xlog(int x) { return 31 - __builtin_clz(x); }

    void init(int *a, int n) {
        for (int i = 0; i < n; i++) {
            st[i][0] = a[i];
        }
        for (int j = 1; (1 << j) <= n; j++) {
            for (int i = 0; i + (1 << j) - 1 < n; i++) {
                st[i][j] = max(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
            }
```

```
        }
    }

    int query(int l, int r) {
        int x = xlog(r - l + 1);
        return max(st[l][x], st[r - (1 << x) + 1][x]);
    }
};

// 二维
struct RMQ {
    int st[MAXN][MAXN][11][11]; // 11 = ((int)log2(MAXN) + 1)

    int xlog(int x) { return 31 - __builtin_clz(x); }

    void init(int n, int m) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                st[i][j][0][0] = a[i][j];
            }
        }
        for (int i = 0; (1 << i) <= n; i++) {
            for (int j = 0; (1 << j) <= m; j++) {
                if (i == 0 && j == 0) continue;
                for (int r = 0; r + (1 << i) - 1 < n; r++) {
                    for (int c = 0; c + (1 << j) - 1 < m; c++) {
                        if (i == 0) {
                            st[r][c][i][j] = max(st[r][c][i][j - 1], st[r][c + (1 << (j - 1))]
[i][j - 1]);
                        } else {
                            st[r][c][i][j] = max(st[r][c][i - 1][j], st[r + (1 << (i - 1))][c]
[i - 1][j]);
                        }
                    }
                }
            }
        }
    }

    int query(int r1, int c1, int r2, int c2) {
        int x = xlog(r2 - r1 + 1);
        int y = xlog(c2 - c1 + 1);
        int m1 = st[r1][c1][x][y];
        int m2 = st[r1][c2 - (1 << y) + 1][x][y];
        int m3 = st[r2 - (1 << x) + 1][c1][x][y];
        int m4 = st[r2 - (1 << x) + 1][c2 - (1 << y) + 1][x][y];
        return max({m1, m2, m3, m4});
    }
};
```

分块

```
// 代码长度没有优势
// 下标从1开始
// 区间加，区间和
struct Node {
    int l, r;
    ll val, lazy;
};
```

```
struct Block {
    int size, b_size; // b_size块 每块大小b_size
    ll *a;
    Node *b;
    int *pos;

    Block(int sz) {
        b_size = ceil(sqrt(sz + 0.5));
        size = b_size * b_size;
        a = new ll[size + 1];
        b = new Node[b_size + 1];
        pos = new int[size + 1];
        for (int i = 1; i <= b_size; i++) {
            b[i].l = (i - 1) * b_size + 1;
            b[i].r = i * b_size;
            b[i].val = b[i].lazy = 0;
        }
        for (int i = 1; i <= size; i++) {
            a[i] = 0;
            pos[i] = (i - 1) / b_size + 1;
        }
    }

    ~Block() {
        delete [] a;
        delete [] b;
        delete [] pos;
    }

    void pushdown(int p) {
        if (!b[p].lazy) return;
        for (int i = b[p].l; i <= b[p].r; i++) {
            a[i] += b[p].lazy;
        }
        b[p].lazy = 0;
    }

    void update(int l, int r, int val) {
        for (int i = pos[l]; i <= pos[r]; i++) {
            if (b[i].l < l || b[i].r > r) {
                pushdown(i);
                for (int j = max(l, b[i].l); j <= min(r, b[i].r); j++) {
                    a[j] += val;
                    b[i].val += val;
                }
            } else {
                b[i].val += (b[i].r - b[i].l + 1) * val;
                b[i].lazy += val;
            }
        }
    }

    ll query(int l, int r) {
        ll ret = 0;
        for (int i = pos[l]; i <= pos[r]; i++) {
            if (b[i].l < l || b[i].r > r) {
                pushdown(i);
                for (int j = max(l, b[i].l); j <= min(r, b[i].r); j++) {
                    ret += a[j];
                }
```

```
            } else {
                ret += b[i].val;
            }
        }
        return ret;
    }
};
```

树状数组

```
// 支持第k大的BIT
// 下标从1开始
// 修改: 单点
// 查询: 区间和
struct Tbit {
    int size;
    ll t[MAXN];

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

    void init(int sz) {
        size = sz + 1;
        memset(t, 0, (sz + 2) * sizeof(ll));
    }

    void add(int pos, ll val) {
        if (pos <= 0) return;
        while (pos <= size) {
            t[pos] += val;
            pos += lowbit(pos);
        }
    }

    ll get(int pos) {
        ll sum = 0;
        while (pos > 0) {
            sum += t[pos];
            pos -= lowbit(pos);
        }
        return sum;
    }

    void update(int pos, ll val) { add(pos, val - query(pos, pos)); }
    ll query(int l, int r) { return get(r) - get(l - 1); }

    int kth(ll k) {
        int p = 0;
        for (int i = 20; i >= 0; i--) {
            int p_ = p + (1 << i);
            if (p_ <= size && t[p_] < k) {
                k -= t[p_];
                p = p_;
            }
        }
        return p + 1;
    }
};

// 修改: 区间加
// 查询: 单点
```

```
struct Tbit {
    int size;
    ll t[MAXN];

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

    void init(int sz) {
        size = sz + 1;
        memset(t, 0, (sz + 2) * sizeof(ll));
    }

    void add(int pos, ll val) {
        if (pos <= 0) return;
        while (pos <= size) {
            t[pos] += val;
            pos += lowbit(pos);
        }
    }

    ll get(int pos) {
        ll sum = 0;
        while (pos > 0) {
            sum += t[pos];
            pos -= lowbit(pos);
        }
        return sum;
    }

    void update(int l, int r, ll val) {
        add(l, val);
        add(r + 1, -val);
    }
};

// 修改: 区间加
// 查询: 区间和
Tbit t1, t2;

void range_add(int l, int r, ll val) {
    t1.add(l, val);
    t2.add(l, l * val);
    t1.add(r + 1, -val);
    t2.add(r + 1, (r + 1) * -val);
}

ll range_sum(int l, int r) {
    return (r + 1) * t1.get(r) - t2.get(r) - l * t1.get(l - 1) + t2.get(l - 1);
}
```

线段树

```
// 下标从1开始
// 修改: 单点
// 查询: RMQ
struct Node {
    int val;
};

struct SegT {
#define lc (p << 1)
```

```
#define rc (p << 1 | 1)
#define mid (pl + pr >> 1)

    int size;
    Node *t;

    SegT(int sz) {
        size = 1;
        while (size < sz) size <<= 1;
        t = new Node[2 * size]();
    }

    ~SegT() {
        delete [] t;
    }

    int ask(int p, int l, int r, int pl, int pr) {
        if (l > pr || r < pl) return -INF;
        if (l <= pl && r >= pr) return t[p].val;
        int vl = ask(lc, l, r, pl, mid);
        int vr = ask(rc, l, r, mid + 1, pr);
        return max(vl, vr);
    }

    void update(int k, int val) {
        int p = size + k - 1;
        t[p].val = val;
        for (p >>= 1; p > 0; p >>= 1) {
            t[p].val = max(t[lc].val, t[rc].val);
        }
    }

    int query(int l, int r) { return ask(1, l, r, 1, size); }

#undef lc
#undef rc
#undef mid
};

// 权值线段树
// 修改: 单点加
// 查询: 第k大
void add(int x, ll val) {
    int p = size + x - 1;
    t[p].val += val;
    for (p >>= 1; p > 0; p >>= 1) {
        t[p].val += val;
    }
}

int ask(int p, ll k, int pl, int pr) {
    if (pl == pr) return pl;
    if (k <= t[lc].val) return ask(lc, k, pl, mid);
    return ask(rc, k - t[lc].val, mid + 1, pr);
}

int query(ll k) { return ask(1, k, 1, size); }

// 修改: 区间加
// 查询: 区间和
```

```
struct Node {
    ll val, lazy;
};

void pushdown(int p, int pl, int pr) {
    if (!t[p].lazy) return;  // 如果是区间赋值，选取一个数据范围外的值
    t[lc].val += t[p].lazy * (mid - pl + 1);
    t[rc].val += t[p].lazy * (pr - mid);
    t[lc].lazy += t[p].lazy;
    t[rc].lazy += t[p].lazy;
    t[p].lazy = 0;
}

ll ask(int p, int l, int r, int pl, int pr) {
    if (l > pr || r < pl) return 0;
    if (l <= pl && r >= pr) return t[p].val;
    pushdown(p, pl, pr);
    ll vl = ask(lc, l, r, pl, mid);
    ll vr = ask(rc, l, r, mid + 1, pr);
    return vl + vr;
}

void modify(int p, int l, int r, int val, int pl, int pr) {
    if (l > pr || r < pl) return;
    if (l <= pl && r >= pr) {
        t[p].val += 1LL * val * (pr - pl + 1);
        t[p].lazy += val;
        return;
    }
    pushdown(p, pl, pr);
    modify(lc, l, r, val, pl, mid);
    modify(rc, l, r, val, mid + 1, pr);
    t[p].val = t[lc].val + t[rc].val;
}

void update(int l, int r, int val) { modify(1, l, r, val, 1, size); }
ll query(int l, int r) { return ask(1, l, r, 1, size); }

// 修改：区间乘混加
// 查询：区间和取模
struct Node {
    ll val, mul, add;
    Node() : val(0), add(0), mul(1) {}
};

void pushdown(int p, int pl, int pr) {
    if (t[p].mul == 1 && t[p].add == 0) return;
    t[lc].val = (t[lc].val * t[p].mul % MOD + (mid - pl + 1) * t[p].add % MOD) % MOD;
    t[rc].val = (t[rc].val * t[p].mul % MOD + (pr - mid) * t[p].add % MOD) % MOD;
    t[lc].mul = t[p].mul * t[lc].mul % MOD;
    t[rc].mul = t[p].mul * t[rc].mul % MOD;
    t[lc].add = (t[lc].add * t[p].mul % MOD + t[p].add) % MOD;
    t[rc].add = (t[rc].add * t[p].mul % MOD + t[p].add) % MOD;
    t[p].mul = 1;
    t[p].add = 0;
}

ll ask(int p, int l, int r, int pl, int pr) {
    if (l > pr || r < pl) return 0;
    if (l <= pl && r >= pr) return t[p].val;
```

```
    pushdown(p, pl, pr);
    ll vl = ask(lc, l, r, pl, mid);
    ll vr = ask(rc, l, r, mid + 1, pr);
    return (vl + vr) % MOD;
}

// x' = ax + b
void modify(int p, int l, int r, int a, int b, int pl, int pr) {
    if (l > pr || r < pl) return;
    if (l <= pl && r >= pr) {
        t[p].val = (t[p].val * a % MOD + 1LL * (pr - pl + 1) * b % MOD) % MOD;
        t[p].mul = t[p].mul * a % MOD;
        t[p].add = (t[p].add * a % MOD + b) % MOD;
        return;
    }
    pushdown(p, pl, pr);
    modify(lc, l, r, a, b, pl, mid);
    modify(rc, l, r, a, b, mid + 1, pr);
    t[p].val = (t[lc].val + t[rc].val) % MOD;
}

void update(int l, int r, int a, int b) { modify(1, l, r, a, b, 1, size); }
ll query(int l, int r) { return ask(1, l, r, 1, size); }
```

主席树

```
struct Node {
    int lc, rc, val;
    Node(int lc = 0, int rc = 0, int val = 0) : lc(lc), rc(rc), val(val) {}
} t[40 * MAXN];

int cnt;

struct FST {
#define mid (pl + pr >> 1)

    int size;
    vector<int> root;

    FST(int sz) {
        size = 1;
        while (size < sz) size <<= 1;
        root.push_back(N(0, 0, 0));
    }

    int N(int lc, int rc, int val) {
        t[cnt] = Node(lc, rc, val);
        return cnt++;
    }

    int ins(int p, int x, int pl, int pr) {
        if (pl > x || pr < x) return p;
        if (pl == pr) return N(0, 0, t[p].val + 1);
        return N(ins(t[p].lc, x, pl, mid), ins(t[p].rc, x, mid + 1, pr), t[p].val + 1);
    }

    int ask(int p1, int p2, int k, int pl, int pr) {
        if (pl == pr) return pl;
        ll vl = t[t[p2].lc].val - t[t[p1].lc].val;
        if (k <= vl) return ask(t[p1].lc, t[p2].lc, k, pl, mid);
```

```
        return ask(t[p1].rc, t[p2].rc, k - vl, mid + 1, pr);
    }

    void add(int x) {
        root.push_back(ins(root.back(), x, 1, size));
    }

    int query(int l, int r, int k) {
        return ask(root[l - 1], root[r], k, 1, size);
    }

#undef mid
};
```

Splay

```
// 正常Splay
struct Node {
    int val, size;
    Node *pa, *lc, *rc;
    Node(int val = 0, Node *pa = nullptr) : val(val), size(1), pa(pa), lc(nullptr),
rc(nullptr) {}
    Node*& c(bool x) { return x ? lc : rc; }
    bool d() { return pa ? this == pa->lc : 0; }
} pool[MAXN], *tail = pool;

struct Splay {
    Node *root;

    Splay() : root(nullptr) {}

    Node* N(int val, Node *pa) {
        return new (tail++) Node(val, pa);
    }

    void pushup(Node *o) {
        o->size = (o->lc ? o->lc->size : 0) + (o->rc ? o->rc->size : 0) + 1;
    }

    void link(Node *x, Node *y, bool d) {
        if (x) x->pa = y;
        if (y) y->c(d) = x;
    }

    void rotate(Node *o) {
        bool dd = o->d();
        Node *x = o->pa, *xx = x->pa, *y = o->c(!dd);
        link(o, xx, x->d());
        link(y, x, dd);
        link(x, o, !dd);
        pushup(x);
        pushup(o);
    }

    void splay(Node *o) {
        for (Node *x = o->pa; x = o->pa, x; rotate(o)) {
            if (x->pa) rotate(o->d() == x->d() ? x : o);
        }
        root = o;
```

```
    }
};
```

图论

链式前向星

```
int ecnt, mp[MAXN];

struct Edge {
    int to, nxt;
    Edge(int to = 0, int nxt = 0) : to(to), nxt(nxt) {}
} es[MAXM];

void mp_init() {
    memset(mp, -1, (n + 2) * sizeof(int));
    ecnt = 0;
}

void mp_link(int u, int v) {
    es[ecnt] = Edge(v, mp[u]);
    mp[u] = ecnt++;
}

for (int i = mp[u]; i != -1; i = es[i].nxt)
```

Dijkstra

```
struct Edge {
    int to, val;
    Edge(int to = 0, int val = 0) : to(to), val(val) {}
};
vector<Edge> G[MAXN];
ll dis[MAXN];

void dijkstra(int s) {
    using pii = pair<ll, int>;
    memset(dis, 0x3f, sizeof(dis));
    priority_queue<pii, vector<pii>, greater<pii> > q;
    dis[s] = 0;
    q.push({0, s});
    while (!q.empty()) {
        pii p = q.top();
        q.pop();
        int u = p.second;
        if (dis[u] < p.first) continue;
        for (int i = 0; i < G[u].size(); i++) {
            int v = G[u][i].to;
            if (dis[v] > dis[u] + G[u][i].val) {
                dis[v] = dis[u] + G[u][i].val;
                q.push({dis[v], v});
            }
        }
    }
}
```

拓扑排序

```
int n, deg[MAXN], dis[MAXN];
vector<int> G[MAXN];

bool topo(vector<int>& ans) {
    queue<int> q;
    for (int i = 1; i <= n; i++) {
        if (deg[i] == 0) {
            q.push(i);
            dis[i] = 1;
        }
    }
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        ans.push_back(u);
        for (int v : G[u]) {
            deg[v]--;
            dis[v] = max(dis[v], dis[u] + 1);
            if (deg[v] == 0) q.push(v);
        }
    }
    return ans.size() == n;
}
```

最小生成树

```
// 前置: 并查集
struct Edge {
    int from, to, val;
    Edge(int from = 0, int to = 0, int val = 0) : from(from), to(to), val(val) {}
};

vector<Edge> es;

ll kruskal() {
    sort(es.begin(), es.end(), [](Edge& x, Edge& y) { return x.val < y.val; });
    iota(pa, pa + n + 1, 0);
    ll ans = 0;
    for (Edge& e : es) {
        if (find(e.from) != find(e.to)) {
            merge(e.from, e.to);
            ans += e.val;
        }
    }
    return ans;
}
```

LCA

```
int dep[MAXN], up[MAXN][22]; // 22 = ((int)log2(MAXN) + 1)

void dfs(int u, int pa) {
    dep[u] = dep[pa] + 1;
    up[u][0] = pa;
    for (int i = 1; i < 22; i++) {
        up[u][i] = up[up[u][i - 1]][i - 1];
    }
    for (int i = 0; i < G[u].size(); i++) {
        if (G[u][i] != pa) {
```

```
            dfs(G[u][i], u);
        }
    }
}

int lca(int u, int v) {
    if (dep[u] > dep[v]) swap(u, v);
    int t = dep[v] - dep[u];
    for (int i = 0; i < 22; i++) {
        if ((t >> i) & 1) v = up[v][i];
    }
    if (u == v) return u;
    for (int i = 21; i >= 0; i--) {
        if (up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    }
    return up[u][0];
}
```

网络流

```
// 最大流
const int INF = 0x7fffffff;

struct Edge {
    int to, cap;
    Edge(int to, int cap) : to(to), cap(cap) {}
};

struct Dinic {
    int n, s, t;
    vector<Edge> es;
    vector<vector<int> > G;
    vector<int> dis, cur;

    Dinic(int n, int s, int t) : n(n), s(s), t(t), G(n + 1), dis(n + 1), cur(n + 1) {}

    void addEdge(int u, int v, int cap) {
        G[u].push_back(es.size());
        es.emplace_back(v, cap);
        G[v].push_back(es.size());
        es.emplace_back(u, 0);
    }

    bool bfs() {
        dis.assign(n + 1, 0);
        queue<int> q;
        q.push(s);
        dis[s] = 1;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i : G[u]) {
                Edge& e = es[i];
                if (!dis[e.to] && e.cap > 0) {
                    dis[e.to] = dis[u] + 1;
                    q.push(e.to);
                }
```

```
            }
        }
        return dis[t];
    }

    int dfs(int u, int cap) {
        if (u == t || cap == 0) return cap;
        int tmp = cap, f;
        for (int& i = cur[u]; i < G[u].size(); i++) {
            Edge& e = es[G[u][i]];
            if (dis[e.to] == dis[u] + 1) {
                f = dfs(e.to, min(cap, e.cap));
                e.cap -= f;
                es[G[u][i] ^ 1].cap += f;
                cap -= f;
                if (cap == 0) break;
            }
        }
        return tmp - cap;
    }

    ll solve() {
        ll flow = 0;
        while (bfs()) {
            cur.assign(n + 1, 0);
            flow += dfs(s, INF);
        }
        return flow;
    }
};

// 最小费用流
const int INF = 0x7fffffff;

struct Edge {
    int from, to, cap, cost;
    Edge(int from, int to, int cap, int cost) : from(from), to(to), cap(cap), cost(cost) {}
};

struct MCMF {
    int n, s, t, flow, cost;
    vector<Edge> es;
    vector<vector<int> > G;
    vector<int> d, p, a;  // dis, prev, add
    deque<bool> in;

    MCMF(int n, int s, int t) : n(n), s(s), t(t), flow(0), cost(0), G(n + 1), p(n + 1), a(n +
1) {}

    void addEdge(int u, int v, int cap, int cost) {
        G[u].push_back(es.size());
        es.emplace_back(u, v, cap, cost);
        G[v].push_back(es.size());
        es.emplace_back(v, u, 0, -cost);
    }

    bool spfa() {
        d.assign(n + 1, INF);
        in.assign(n + 1, false);
        d[s] = 0;
```

```
        in[s] = 1;
        a[s] = INF;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            in[u] = false;
            for (int& i : G[u]) {
                Edge& e = es[i];
                if (e.cap && d[e.to] > d[u] + e.cost) {
                    d[e.to] = d[u] + e.cost;
                    p[e.to] = i;
                    a[e.to] = min(a[u], e.cap);
                    if (!in[e.to]) {
                        q.push(e.to);
                        in[e.to] = true;
                    }
                }
            }
        }
        return d[t] != INF;
    }

    void solve() {
        while (spfa()) {
            flow += a[t];
            cost += a[t] * d[t];
            int u = t;
            while (u != s) {
                es[p[u]].cap -= a[t];
                es[p[u] ^ 1].cap += a[t];
                u = es[p[u]].from;
            }
        }
    }
};
```

树链剖分

```
// 点权
vector<int> G[MAXN];
int pa[MAXN], sz[MAXN], dep[MAXN], dfn[MAXN], maxc[MAXN], top[MAXN];

void dfs1(int u) {
    sz[u] = 1;
    maxc[u] = -1;
    int maxs = 0;
    for (int& v : G[u]) {
        if (v != pa[u]) {
            pa[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            sz[u] += sz[v];
            if (updmax(maxs, sz[v])) maxc[u] = v;
        }
    }
}

void dfs2(int u, int tp) {
```

```
    static int cnt = 0;
    top[u] = tp;
    dfn[u] = ++cnt;
    if (maxc[u] != -1) dfs2(maxc[u], tp);
    for (int& v : G[u]) {
        if (v != pa[u] && v != maxc[u]) {
            dfs2(v, v);
        }
    }
}

void init() {
    dep[1] = 1;
    dfs1(1);
    dfs2(1, 1);
}

ll go(int u, int v) {
    int uu = top[u], vv = top[v];
    ll res = 0;
    while (uu != vv) {
        if (dep[uu] < dep[vv]) {
            swap(u, v);
            swap(uu, vv);
        }
        res += segt.query(dfn[uu], dfn[u]);
        u = pa[uu];
        uu = top[u];
    }
    if (dep[u] > dep[v]) swap(u, v);
    res += segt.query(dfn[u], dfn[v]);
    return res;
}
```

字符串

哈希

```
// open hack不要用哈希
using ull = unsigned long long;

const int x = 135, p1 = 1e9 + 7, p2 = 1e9 + 9;

int n;
char s[MAXN];
ull xp1[MAXN], xp2[MAXN], h[MAXN];

void init_xp() {
    xp1[0] = xp2[0] = 1;
    for (int i = 1; i < MAXN; i++) {
        xp1[i] = xp1[i - 1] * x % p1;
        xp2[i] = xp2[i - 1] * x % p2;
    }
}

void init_hash() {
    ull res1 = 0, res2 = 0;
    h[n + 1] = 0;
    for (int i = n; i >= 0; i--) {
```

```
        res1 = (res1 * x + s[i]) % p1;
        res2 = (res2 * x + s[i]) % p2;
        h[i] = (res1 << 32) | res2;
    }
}

ull get_hash(int l, int r) {
    r++;
    int len = r - l;
    unsigned int mask32 = ~(0u);
    ull l1 = h[l] >> 32, r1 = h[r] >> 32;
    ull l2 = h[l] & mask32, r2 = h[r] & mask32;
    ull res1 = (l1 - r1 * xp1[len] % p1 + p1) % p1;
    ull res2 = (l2 - r2 * xp2[len] % p2 + p2) % p2;
    return (res1 << 32) | res2;
}
```

Manacher

```
// "aba" => "#a#b#a#"
string make(string& s) {
    string t = "#";
    for (int i = 0; i < s.size(); i++) {
        t.push_back(s[i]);
        t.push_back('#');
    }
    return t;
}

void manacher(string& s, vector<int>& d) {
    int n = s.size();
    d.resize(n);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d[l + r - i], r - i);
        while (i - k >= 0 && i + k < n && s[i - k] == s[i + k]) k++;
        d[i] = --k;
        if (i + k > r) {
            l = i - k;
            r = i + k;
        }
    }
}
```

KMP

```
// 前缀函数（每一个前缀的最长公共前后缀）
void get_pi(const string& s, vector<int>& a) {
    int n = s.size(), j = 0;
    a.resize(n);
    for (int i = 1; i < n; i++) {
        while (j && s[j] != s[i]) j = a[j - 1];
        if (s[j] == s[i]) j++;
        a[i] = j;
    }
}

void kmp(const string& s, vector<int>& a, const string& t) {
    int j = 0;
    for (int i = 0; i < t.size(); i++) {
```

```
        while (j && s[j] != t[i]) j = a[j - 1];
        if (s[j] == t[i]) j++;
        if (j == s.size()) {
            // ...
            j = a[j - 1]; // 允许重叠匹配 j = 0 不允许
        }
    }
}

// Z函数（每一个后缀和该字符串的最长公共前缀）
void get_z(const string& s, vector<int>& z) {
    int n = s.size(), l = 0, r = 0;
    z.resize(n);
    for (int i = 1; i < n; i++) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) z[i]++;
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
}
```

Trie

```
// 01 Trie
struct Trie {
    int t[31 * MAXN][2], sz;

    void init() {
        memset(t, 0, 2 * (sz + 2) * sizeof(int));
        sz = 1;
    }

    void insert(int x) {
        int p = 0;
        for (int i = 30; i >= 0; i--) {
            bool d = (x >> i) & 1;
            if (!t[p][d]) t[p][d] = sz++;
            p = t[p][d];
        }
    }
};

// 正常Trie
struct Trie {
    int t[MAXN][26], sz, cnt[MAXN];

    void init() {
        memset(t, 0, 26 * (sz + 2) * sizeof(int));
        memset(cnt, 0, (sz + 2) * sizeof(int));
        sz = 1;
    }

    void insert(const string& s) {
        int p = 0;
        for (char c : s) {
            int d = c - 'a';
            if (!t[p][d]) t[p][d] = sz++;
            p = t[p][d];
```

```
        }
        cnt[p]++;
    }
};
```

AC 自动机

```
struct ACA {
    int t[MAXN][26], sz, fail[MAXN], nxt[MAXN], cnt[MAXN];

    void init() {
        memset(t, 0, 26 * (sz + 2) * sizeof(int));
        memset(fail, 0, (sz + 2) * sizeof(int));
        memset(nxt, 0, (sz + 2) * sizeof(int));
        memset(cnt, 0, (sz + 2) * sizeof(int));
        sz = 1;
    }

    void insert(const string& s) {
        int p = 0;
        for (char c : s) {
            int d = c - 'a';
            if (!t[p][d]) t[p][d] = sz++;
            p = t[p][d];
        }
        cnt[p]++;
    }

    void build() {
        queue<int> q;
        for (int i = 0; i < 26; i++) {
            if (t[0][i]) q.push(t[0][i]);
        }
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++) {
                int& v = t[u][i];
                if (v) {
                    fail[v] = t[fail[u]][i];
                    nxt[v] = cnt[fail[v]] ? fail[v] : nxt[fail[v]];
                    q.push(v);
                } else {
                    v = t[fail[u]][i];
                }
            }
        }
    }
};
```

数学

GCD & LCM

```
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
```

快速幂 & 快速乘

```
// 注意 b = 0, MOD = 1 的情况
ll powMod(ll a, ll b) {
    ll ans = 1;
    for (a %= MOD; b; b >>= 1) {
        if (b & 1) ans = ans * a % MOD;
        a = a * a % MOD;
    }
    return ans;
}

// 模数爆int时使用
ll mul(ll a, ll b) {
    ll ans = 0;
    for (a %= MOD; b; b >>= 1) {
        if (b & 1) ans = (ans + a) % MOD;
        a = (a << 1) % MOD;
    }
    return ans;
}

// O(1)
ll mul(ll a, ll b) {
    return (ll)(__int128(a) * b % MOD);
}
```

矩阵快速幂

```
const int MAT_SZ = 3;

struct Mat {
    ll m[MAT_SZ][MAT_SZ] = {0};
    ll * operator [] (int i) { return m[i]; }
    void one() { for (int i = 0; i < MAT_SZ; i++) m[i][i] = 1; }
};

Mat mul(Mat &a, Mat &b) {
    Mat ans;
    for (int i = 0; i < MAT_SZ; i++)
        for (int j = 0; j < MAT_SZ; j++)
            if (a[i][j])
                for (int k = 0; k < MAT_SZ; k++)
                    ans[i][k] = (ans[i][k] + a[i][j] * b[j][k]) % MOD;
    return ans;
}

Mat pow(Mat &a, ll b) {
    Mat ans;
    ans.one();
    while (b) {
        if (b & 1) ans = mul(a, ans);
        b >>= 1;
        a = mul(a, a);
    }
    return ans;
}
```

素数判断

```
bool isPrime(int x) {
    if (x < 2) return false;
    for (int i = 2; i * i <= x; i++) if (x % i == 0) return false;
    return true;
}

// O(logn)
// 前置：快速幂、快速乘
// int范围只需检查2, 7, 61
bool Rabin_Miller(ll p, ll a) {
    if (p == 2) return 1;
    if (p & 1 == 0 || p == 1) return 0;
    ll d = p - 1;
    while (!(d & 1)) d >>= 1;
    ll m = powMod(a, d, p);
    if (m == 1) return 1;
    while (d < p) {
        if (m == p - 1) return 1;
        d <<= 1;
        m = mul(m, m, p);
    }
    return 0;
}

bool isPrime(ll x) {
    if (x == 3 || x == 5) return 1;
    static ll prime[7] = {2, 307, 7681, 36061, 555097, 4811057, 1007281591};
    for (int i = 0; i < 7; i++) {
        if (x == prime[i]) return 1;
        if (!Rabin_Miller(x, prime[i])) return 0;
    }
    return 1;
}
```

线性筛

```
// 注意 0 和 1 不是素数
bool vis[MAXN];
int prime[MAXN];

void get_prime() {
    int tot = 0;
    for (int i = 2; i < MAXN - 5; i++) {
        if (!vis[i]) prime[tot++] = i;
        for (int j = 0; j < tot; j++) {
            int d = i * prime[j];
            if (d >= MAXN - 5) break;
            vis[d] = true;
            if (i % prime[j] == 0) break;
        }
    }
}

// 最小素因子
bool vis[MAXN];
int spf[MAXN], prime[MAXN];

void get_spf() {
    int tot = 0;
    for (int i = 2; i < MAXN - 5; i++) {
```

```
        if (!vis[i]) {
            prime[tot++] = i;
            spf[i] = i;
        }
        for (int j = 0; j < tot; j++) {
            int d = i * prime[j];
            if (d >= MAXN - 5) break;
            vis[d] = true;
            spf[d] = prime[j];
            if (i % prime[j] == 0) break;
        }
    }
}

// 欧拉函数
bool vis[MAXN];
int phi[MAXN], prime[MAXN];

void get_phi() {
    int tot = 0;
    phi[1] = 1;
    for (int i = 2; i < MAXN - 5; i++) {
        if (!vis[i]) {
            prime[tot++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; j < tot; j++) {
            int d = i * prime[j];
            if (d >= MAXN - 5) break;
            vis[d] = true;
            if (i % prime[j] == 0) {
                phi[d] = phi[i] * prime[j];
                break;
            }
            else phi[d] = phi[i] * (prime[j] - 1);
        }
    }
}

// 莫比乌斯函数
bool vis[MAXN];
int mu[MAXN], prime[MAXN];

void get_mu() {
    int tot = 0;
    mu[1] = 1;
    for (int i = 2; i < MAXN - 5; i++) {
        if (!vis[i]) {
            prime[tot++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < tot; j++) {
            int d = i * prime[j];
            if (d >= MAXN - 5) break;
            vis[d] = true;
            if (i % prime[j] == 0) {
                mu[d] = 0;
                break;
            }
            else mu[d] = -mu[i];
```

```
        }
    }
}
```

找因数

```
// O(sqrt(n))
void getf(int x, vector<int> &v) {
    for (int i = 1; i * i <= x; i++) {
        if (x % i == 0) {
            v.push_back(i);
            if (x / i != i) v.push_back(x / i);
        }
    }
    sort(v.begin(), v.end());
}
```

找质因数

```
// O(sqrt(n))，无重复
void getf(int x, vector<int> &v) {
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) {
            v.push_back(i);
            while (x % i == 0) x /= i;
        }
    }
    if (x != 1) v.push_back(x);
}

// O(sqrt(n))，有重复
void getf(int x, vector<int> &v) {
    for (int i = 2; i * i <= x; i++) {
        while (x % i == 0) {
            v.push_back(i);
            x /= i;
        }
    }
    if (x != 1) v.push_back(x);
}

// 前置：线性筛
// O(logn)，无重复
void getf(int x, vector<int> &v) {
    while (x > 1) {
        int p = spf[x];
        v.push_back(p);
        while (x % p == 0) x /= p;
    }
}

// O(logn)，有重复
void getf(int x, vector<int> &v) {
    while (x > 1) {
        int p = spf[x];
        while (x % p == 0) {
            v.push_back(p);
            x /= p;
        }
```

```
    }
}
```

欧拉函数

```
// 前置：找质因数（无重复）
int phi(int x) {
    int ret = x;
    vector<int> v;
    getf(x, v);
    for (int f : v) ret = ret / f * (f - 1);
    return ret;
}

// O(nloglogn)
int phi[MAXN];

void get_phi() {
    phi[1] = 1;
    for (int i = 2; i < MAXN - 5; i++) {
        if (!phi[i]) {
            for (int j = i; j < MAXN - 5; j += i) {
                if (!phi[j]) phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
        }
    }
}
```

EXGCD

```
// ax + by = gcd(a, b)
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

逆元

```
ll inv(ll x) { return powMod(x, MOD - 2); }

// EXGCD
// gcd(a, p) = 1时有逆元
ll inv(ll a, ll p) {
    ll x, y;
    ll d = exgcd(a, p, x, y);
    if (d == 1) return (x % p + p) % p;
    return -1;
}

// 逆元打表
ll inv[MAXN];
```

```
void initInv() {
    inv[1] = 1;
    for (int i = 2; i < MAXN - 5; i++) {
        inv[i] = 1LL * (MOD - MOD / i) * inv[MOD % i] % MOD;
    }
}
```

组合数

```
// 组合数打表
ll C[MAXN][MAXN];

void initC() {
    C[0][0] = 1;
    for (int i = 1; i < MAXN - 5; i++) {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++) {
            C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
        }
    }
}

// 快速组合数取模
// MAXN开2倍上限
ll fac[MAXN], ifac[MAXN];

void initInv() {
    fac[0] = 1;
    for (int i = 1; i < MAXN; i++) {
        fac[i] = fac[i - 1] * i % MOD;
    }
    ifac[MAXN - 1] = powMod(fac[MAXN - 1], MOD - 2);
    for (int i = MAXN - 2; i >= 0; i--) {
        ifac[i] = ifac[i + 1] * (i + 1);
        ifac[i] %= MOD;
    }
}

ll C(int n, int m) {
    if (n < m || m < 0) return 0;
    return fac[n] * ifac[m] % MOD * ifac[n - m] % MOD;
}

// Lucas
ll C(ll n, ll m) {
    if (n < m || m < 0) return 0;
    if (n < MOD && m < MOD) return fac[n] * ifac[m] % MOD * ifac[n - m] % MOD;
    return C(n / MOD, m / MOD) * C(n % MOD, m % MOD) % MOD;
}
```

康托展开

```
// 需要预处理阶乘
int cantor(vector<int>& s) {
    int n = s.size(), ans = 0;
    for (int i = 0; i < n - 1; i++) {
        int cnt = 0;
        for (int j = i + 1; j < n; j++) {
            if (s[j] < s[i]) cnt++;
```

```
        }
        ans += cnt * fac[n - i - 1];
    }
    return ans + 1;
}

vector<int> inv_cantor(int x, int n) {
    x--;
    vector<int> ans(n), rk(n);
    iota(rk.begin(), rk.end(), 1);
    for (int i = 0; i < n; i++) {
        int t = x / fac[n - i - 1];
        x %= fac[n - i - 1];
        ans[i] = rk[t];
        for (int j = t; rk[j] < n; j++) {
            rk[j] = rk[j + 1];
        }
    }
    return ans;
}
```

自适应Simpson积分

```
double simpson(double l, double r) {
    double c = (l + r) / 2;
    return (f(l) + 4 * f(c) + f(r)) * (r - l) / 6;
}

double asr(double l, double r, double eps, double S) {
    double mid = (l + r) / 2;
    double L = simpson(l, mid), R = simpson(mid, r);
    if (fabs(L + R - S) < 15 * eps) return L + R + (L + R - S) / 15;
    return asr(l, mid, eps / 2, L) + asr(mid, r, eps / 2, R);
}

double asr(double l, double r) { return asr(l, r, EPS, simpson(l, r)); }
```

拉格朗日插值

```
vector<double> La(vector<pair<double, double> > v) {
    int n = v.size(), t;
    vector<double> ret(n);
    double p, q;
    for (int i = 0; i < n; i++) {
        p = v[i].second;
        for (int j = 0; j < n; j++) {
            p /= (i == j) ? 1 : (v[i].first - v[j].first);
        }
        for (int j = 0; j < (1 << n); j++) {
            q = 1, t = 0;
            for (int k = 0; k < n; k++) {
                if (i == k) continue;
                if ((j >> k) & 1) q *= -v[k].first;
                else t++;
            }
            ret[t] += p * q / 2;
        }
    }
```

```
    return ret;
}
```

计算几何

二维几何基础

```
#define y1 qwq

const double PI = acos(-1);
const double EPS = 1e-8;

int sgn(double x) { return x < -EPS ? -1 : x > EPS; }

// 不要直接使用sgn
bool eq(double x, double y) { return sgn(x - y) == 0; }
bool lt(double x, double y) { return sgn(x - y) < 0; }
bool gt(double x, double y) { return sgn(x - y) > 0; }
bool leq(double x, double y) { return sgn(x - y) <= 0; }
bool geq(double x, double y) { return sgn(x - y) >= 0; }

struct V {
    double x, y;
    V(double x = 0, double y = 0) : x(x), y(y) {}
    V(const V& a, const V& b) : x(b.x - a.x), y(b.y - a.y) {}
    V operator + (const V &b) const { return V(x + b.x, y + b.y); }
    V operator - (const V &b) const { return V(x - b.x, y - b.y); }
    V operator * (double k) const { return V(x * k, y * k); }
    V operator / (double k) const { return V(x / k, y / k); }
    double len() const { return hypot(x, y); }
    double len2() const { return x * x + y * y; }
};

ostream& operator << (ostream& os, const V& p) { return os << "(" << p.x << ", " << p.y <<
")"; }
istream& operator >> (istream& is, V& p) { return is >> p.x >> p.y; }

double dist(const V& a, const V& b) { return (b - a).len(); }
double dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
double det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
double cross(const V& s, const V& t, const V& o) { return det(V(o, s), V(o, t)); }

// 逆时针旋转 r 弧度
V rot(const V& p, double r) {
    return V(p.x * cos(r) - p.y * sin(r), p.x * sin(r) + p.y * cos(r));
}
V rot_ccw90(const V& p) { return V(-p.y, p.x); }
V rot_cw90(const V& p) { return V(p.y, -p.x); }

// 点在线段上 leq(dot(...), 0) 包含端点 lt(dot(...), 0) 则不包含
bool p_on_seg(const V& p, const V& a, const V& b) {
    return eq(det(p - a, b - a), 0) && leq(dot(p - a, p - b), 0);
}

// 点在射线上 geq(dot(...), 0) 包含端点 gt(dot(...), 0) 则不包含
bool p_on_ray(const V& p, const V& a, const V& b) {
    return eq(det(p - a, b - a), 0) && geq(dot(p - a, b - a), 0);
}
```

```
// 点到直线距离
double dist_to_line(const V& p, const V& a, const V& b) {
    return abs(cross(a, b, p) / dist(a, b));
}

// 点到线段距离
double dist_to_seg(const V& p, const V& a, const V& b) {
    if (lt(dot(b - a, p - a), 0)) return dist(p, a);
    if (lt(dot(a - b, p - b), 0)) return dist(p, b);
    return dist_to_line(p, a, b);
}

// 求直线交点
V intersect(const V& a, const V& b, const V& c, const V& d) {
    double s1 = cross(c, d, a), s2 = cross(c, d, b);
    return (a * s2 - b * s1) / (s2 - s1);
}
```

多边形

```
// 多边形面积
double area(const vector<V>& s) {
    double ret = 0;
    for (int i = 0; i < s.size(); i++) {
        ret += det(s[i], s[(i + 1) % s.size()]);
    }
    return ret / 2;
}

// 多边形重心
V centroid(const vector<V>& s) {
    V c;
    for (int i = 0; i < s.size(); i++) {
        c = c + (s[i] + s[(i + 1) % s.size()]) * det(s[i], s[(i + 1) % s.size()]);
    }
    return c / 6.0 / area(s);
}

// 点是否在多边形中
// 1 inside 0 on border -1 outside
int inside(const vector<V>& s, const V& p) {
    int cnt = 0;
    for (int i = 0; i < s.size(); i++) {
        V a = s[i], b = s[(i + 1) % s.size()];
        if (p_on_seg(p, a, b)) return 0;
        if (leq(a.y, b.y)) swap(a, b);
        if (gt(p.y, a.y)) continue;
        if (leq(p.y, b.y)) continue;
        cnt += gt(cross(b, a, p), 0);
    }
    return (cnt & 1) ? 1 : -1;
}

// 构建凸包 点不可以重复
// lt(cross(...), 0) 边上可以有点 leq(cross(...), 0) 则不能
// 会改变输入点的顺序
vector<V> convex_hull(vector<V>& s) {
    // assert(s.size() >= 3);
    sort(s.begin(), s.end(), [](V &a, V &b) { return eq(a.x, b.x) ? lt(a.y, b.y) : lt(a.x,
b.x); });
```

```
    vector<V> ret(2 * s.size());
    int sz = 0;
    for (int i = 0; i < s.size(); i++) {
        while (sz > 1 && leq(cross(ret[sz - 1], s[i], ret[sz - 2]), 0)) sz--;
        ret[sz++] = s[i];
    }
    int k = sz;
    for (int i = s.size() - 2; i >= 0; i--) {
        while (sz > k && leq(cross(ret[sz - 1], s[i], ret[sz - 2]), 0)) sz--;
        ret[sz++] = s[i];
    }
    ret.resize(sz - (s.size() > 1));
    return ret;
}

// 多边形是否为凸包
bool is_convex(const vector<V>& s) {
    for (int i = 0; i < s.size(); i++) {
        if (lt(cross(s[(i + 1) % s.size()], s[(i + 2) % s.size()], s[i]), 0)) return false;
    }
    return true;
}

// 点是否在凸包中
// 1 inside 0 on border -1 outside
int inside(const vector<V>& s, const V& p) {
    for (int i = 0; i < s.size(); i++) {
        if (lt(cross(s[i], s[(i + 1) % s.size()], p), 0)) return -1;
        if (p_on_seg(p, s[i], s[(i + 1) % s.size()])) return 0;
    }
    return 1;
}
```

圆

```
struct C {
    V o;
    double r;
    C(double x = 0, double y = 0, double r = 0) : o(x, y), r(r) {}
    C(const V& o, double r) : o(o), r(r) {}
};
```

三维几何

杂项

updmax/min

```
template<typename T> inline bool updmax(T &a, T b) { return a < b ? a = b, 1 : 0; }
template<typename T> inline bool updmin(T &a, T b) { return a > b ? a = b, 1 : 0; }
```

二分答案

```
// 二分闭区间[l, r]
// 可行下界
while (l < r) {
    mid = (l + r) / 2;
```

```
    if (check(mid)) r = mid;
    else l = mid + 1;
}

// 可行上界
while (l < r) {
    mid = (l + r + 1) / 2;
    if (check(mid)) l = mid;
    else r = mid - 1;
}
```

三分

```
// 实数范围
double l, r, mid1, mid2;
for (int i = 0; i < 75; i++) {
    mid1 = (l * 5 + r * 4) / 9;
    mid2 = (l * 4 + r * 5) / 9;
    if (f(mid1) > f(mid2)) r = mid2; // 单峰函数取'>'号，单谷函数取'<'号
    else l = mid1;
}

// 整数范围
int l, r, mid1, mid2;
while (l < r - 2) {
    mid1 = (l + r) / 2;
    mid2 = mid1 + 1;
    if (f(mid1) > f(mid2)) r = mid2; // 单峰函数取'>'号，单谷函数取'<'号
    else l = mid1;
}
int maxval = f(l), ans = l;
for (int i = l + 1; i <= r; i++) {
    if (updmax(maxval, f(i))) ans = i;
}
```

日期

```
int date_to_int(int y, int m, int d) {
    return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

void int_to_date(int jd, int &y, int &m, int &d) {
    int x, n, i, j;

    x = jd + 68569;
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}
```

子集枚举

```cpp
// 枚举真子集
for (int t = (x - 1) & x; t; t = (t - 1) & x)

// 枚举大小为 k 的子集
// 注意 k 不能为 0
void subset(int k, int n) {
    int t = (1 << k) - 1;
    while (t < (1 << n)) {
        // do something
        int x = t & -t, y = t + x;
        t = ((t & ~y) / x >> 1) | y;
    }
}
```

表达式求值

```python
print(input()) # Python2
print(eval(input())) # Python3
```

对拍

- *unix

```bash
#!/bin/bash
cd "$(dirname "${BASH_SOURCE[0]}")"

g++ gen.cpp -o gen -O2 -std=c++11
g++ my.cpp -o my -O2 -std=c++11
g++ std.cpp -o std -O2 -std=c++11

while true
do
    ./gen > in.txt
    ./std < in.txt > stdout.txt
    ./my < in.txt > myout.txt

    if test $? -ne 0
    then
        printf "RE\n"
        exit 0
    fi

    if diff stdout.txt myout.txt
    then
        printf "AC\n"
    else
        printf "WA\n"
        exit 0
    fi
done
```

- Windows

```bat
@echo off

g++ gen.cpp -o gen.exe -O2 -std=c++11
g++ my.cpp -o my.exe -O2 -std=c++11
```

```
g++ std.cpp -o std.exe -O2 -std=c++11

:loop
    gen.exe > in.txt
    std.exe < in.txt > stdout.txt
    my.exe < in.txt > myout.txt
    if errorlevel 1 (
        echo RE
        pause
        exit
    )
    fc stdout.txt myout.txt
    if errorlevel 1 (
        echo WA
        pause
        exit
    )
goto loop
```

pb_ds

```cpp
// 平衡树
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<class T>
using rank_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template<class Key, class T>
using rank_map = tree<Key, T, less<Key>, rb_tree_tag, tree_order_statistics_node_update>;

// 优先队列
#include <ext/pb_ds/priority_queue.hpp>
using namespace __gnu_pbds;
template<class T, class Cmp = less<T> >
using pair_heap = __gnu_pbds::priority_queue<T, Cmp>;
```