代码库

上海交通大学

2015 年 10 月 11 日

目录

1 数论

1.1 快速求逆元

返回结果：

$$x^{-1}(mod)$$

使用条件：$x \in [0, mod)$ 并且 x 与 mod 互质

```
long long inverse(const long long &x, const long long &mod) {
    if (x == 1) {
        return 1;
    } else {
        return (mod - mod / x) * inverse(mod % x, mod) % mod;
    }
}
```

1.2 扩展欧几里德算法

返回结果：

$$ax + by = gcd(a, b)$$

时间复杂度：$\mathcal{O}(nlogn)$

```
void solve(const long long &a, const long long &b, long long &x, long long &y) {
    if (b == 0) {
        x = 1;
        y = 0;
    } else {
        solve(b, a % b, x, y);
        x -= a / b * y;
        std::swap(x, y);
    }
}
```

1.3 中国剩余定理

返回结果：

$$x \equiv r_i(mod\ p_i)\ (0 \leq i < n)$$

使用条件：p_i 无需两两互质

时间复杂度：$\mathcal{O}(nlogn)$

```
bool solve(int n, std::pair<long long, long long> input[],
                  std::pair<long long, long long> &output) {
    output = std::make_pair(1, 1);
```

```
    for (int i = 0; i < n; ++i) {
        long long number, useless;
        euclid(output.second, input[i].second, number, useless);
        long long divisor = std::__gcd(output.second, input[i].second);
        if ((input[i].first - output.first) % divisor) {
            return false;
        }
        number *= (input[i].first - output.first) / divisor;
        fix(number, input[i].second);
        output.first += output.second * number;
        output.second *= input[i].second / divisor;
        fix(output.first, output.second);
    }
    return true;
}
```

1.4 Miller Rabin 素数测试

```
const int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

bool check(const long long &prime, const long long &base) {
    long long number = prime - 1;
    for (; ~number & 1; number >>= 1);
    long long result = power_mod(base, number, prime);
    for (; number != prime - 1 && result != 1 && result != prime - 1; number <<= 1) {
        result = multiply_mod(result, result, prime);
    }
    return result == prime - 1 || (number & 1) == 1;
}

bool miller_rabin(const long long &number) {
    if (number < 2) {
        return false;
    }
    if (number < 4) {
        return true;
    }
    if (~number & 1) {
        return false;
    }
    for (int i = 0; i < 12 && BASE[i] < number; ++i) {
        if (!check(number, BASE[i])) {
            return false;
```

```
        }
    }
    return true;
}
```

1.5 Pollard Rho 大数分解

时间复杂度：$\mathcal{O}(n^{1/4})$

```
long long pollard_rho(const long long &number, const long long &seed) {
    long long x = rand() % (number - 1) + 1, y = x;
    for (int head = 1, tail = 2; ; ) {
        x = multiply_mod(x, x, number);
        x = add_mod(x, seed, number);
        if (x == y) {
            return number;
        }
        long long answer = std::__gcd(abs(x - y), number);
        if (answer > 1 && answer < number) {
            return answer;
        }
        if (++head == tail) {
            y = x;
            tail <<= 1;
        }
    }
}

void factorize(const long long &number, std::vector<long long> &divisor) {
    if (number > 1) {
        if (miller_rabin(number)) {
            divisor.push_back(number);
        } else {
            long long factor = number;
            for (; factor >= number;
                   factor = pollard_rho(number, rand() % (number - 1) + 1));
            factorize(number / factor, divisor);
            factorize(factor, divisor);
        }
    }
}
```

1.6 快速数论变换

返回结果：

$$c_i = \sum_{0\le j\le i} a_j \cdot b_{i-j} (mod)\ (0 \le i < n)$$

使用说明：$magic$ 是 mod 的原根

时间复杂度：$\mathcal{O}(nlogn)$

```
void solve(long long number[], int length, int type) {
    for (int i = 1, j = 0; i < length - 1; ++i) {
        for (int k = length; j ^= k >>= 1, ~j & k; );
        if (i < j) {
            std::swap(number[i], number[j]);
        }
    }
    long long unit_p0;
    for (int turn = 0; (1 << turn) < length; ++turn) {
        int step = 1 << turn, step2 = step << 1;
        if (type == 1) {
            unit_p0 = power_mod(MAGIC, (MOD - 1) / step2, MOD);
        } else {
            unit_p0 = power_mod(MAGIC, MOD - 1 - (MOD - 1) / step2, MOD);
        }
        for (int i = 0; i < length; i += step2) {
            long long unit = 1;
            for (int j = 0; j < step; ++j) {
                long long &number1 = number[i + j + step];
                long long &number2 = number[i + j];
                long long delta = unit * number1 % MOD;
                number1 = (number2 - delta + MOD) % MOD;
                number2 = (number2 + delta) % MOD;
                unit = unit * unit_p0 % MOD;
            }
        }
    }
}

void multiply() {
    for (; lowbit(length) != length; ++length);
    solve(number1, length, 1);
    solve(number2, length, 1);
    for (int i = 0; i < length; ++i) {
        number[i] = number1[i] * number2[i] % MOD;
    }
```

```
    solve(number, length, -1);
    for (int i = 0; i < length; ++i) {
        answer[i] = number[i] * power_mod(length, MOD - 2, MOD) % MOD;
    }
}
```

1.7 原根

1.8 离散对数

1.9 离散平方根

1.10 佩尔方程求解

1.11 直线下整点个数

返回结果：

$$\sum_{0 \le i < n} \lfloor \frac{a + b \cdot i}{m} \rfloor$$

使用条件：$n, m > 0$，$a, b \ge 0$

时间复杂度：$\mathcal{O}(nlogn)$

```
long long solve(const long long &n, const long long &a,
                const long long &b, const long long &m) {
    if (b == 0) {
        return n * (a / m);
    }
    if (a >= m) {
        return n * (a / m) + solve(n, a % m, b, m);
    }
    if (b >= m) {
        return (n - 1) * n / 2 * (b / m) + solve(n, a, b % m, m);
    }
    return solve((a + b * n) / m, (a + b * n) % m, m, b);
}
```

2 数值

2.1 高斯消元

2.2 快速傅立叶变换

返回结果：

$$c_i = \sum_{0 \le j \le i} a_j \cdot b_{i-j} \ (0 \le i < n)$$

时间复杂度：$\mathcal{O}(nlogn)$

```
void solve(Complex number[], int length, int type) {
    for (int i = 1, j = 0; i < length - 1; ++i) {
        for (int k = length; j ^= k >>= 1, ~j & k; );
        if (i < j) {
            std::swap(number[i], number[j]);
        }
    }
    Complex unit_p0;
    for (int turn = 0; (1 << turn) < length; ++turn) {
        int step = 1 << turn, step2 = step << 1;
        double p0 = PI / step * type;
        sincos(p0, &unit_p0.imag(), &unit_p0.real());
        for (int i = 0; i < length; i += step2) {
            Complex unit = 1;
            for (int j = 0; j < step; ++j) {
                Complex &number1 = number[i + j + step];
                Complex &number2 = number[i + j];
                Complex delta = unit * number1;
                number1 = number2 - delta;
                number2 = number2 + delta;
                unit = unit * unit_p0;
            }
        }
    }
}

void multiply() {
    for (; lowbit(length) != length; ++length);
    solve(number1, length, 1);
    solve(number2, length, 1);
    for (int i = 0; i < length; ++i) {
        number[i] = number1[i] * number2[i];
    }
    solve(number, length, -1);
    for (int i = 0; i < length; ++i) {
        answer[i] = (int)(number[i].real() / length + 0.5);
    }
}
```

2.3 单纯形法求解线性规划

返回结果：

$$max\{c_{1\times m}\cdot x_{m\times 1}\mid x_{m\times 1}\geq 0_{m\times 1}, a_{n\times m}\cdot x_{m\times 1}\leq b_{n\times 1}\}$$

```cpp
std::vector<double> solve(const std::vector<std::vector<double> > &a,
                          const std::vector<double> &b, const std::vector<double> &c) {
    int n = (int)a.size(), m = (int)a[0].size() + 1;
    std::vector<std::vector<double> > value(n + 2, std::vector<double>(m + 1));
    std::vector<int> index(n + m);
    int r = n, s = m - 1;
    for (int i = 0; i < n + m; ++i) {
        index[i] = i;
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m - 1; ++j) {
            value[i][j] = -a[i][j];
        }
        value[i][m - 1] = 1;
        value[i][m] = b[i];
        if (value[r][m] > value[i][m]) {
            r = i;
        }
    }
    for (int j = 0; j < m - 1; ++j) {
        value[n][j] = c[j];
    }
    value[n + 1][m - 1] = -1;
    for (double number; ; ) {
        if (r < n) {
            std::swap(index[s], index[r + m]);
            value[r][s] = 1 / value[r][s];
            for (int j = 0; j <= m; ++j) {
                if (j != s) {
                    value[r][j] *= -value[r][s];
                }
            }
            for (int i = 0; i <= n + 1; ++i) {
                if (i != r) {
                    for (int j = 0; j <= m; ++j) {
                        if (j != s) {
                            value[i][j] += value[r][j] * value[i][s];
                        }
                    }
```

```
                    value[i][s] *= value[r][s];
                }
            }
        }
        r = s = -1;
        for (int j = 0; j < m; ++j) {
            if (s < 0 || index[s] > index[j]) {
                if (value[n + 1][j] > eps || value[n + 1][j] > -eps && value[n][j] > eps) {
                    s = j;
                }
            }
        }
        if (s < 0) {
            break;
        }
        for (int i = 0; i < n; ++i) {
            if (value[i][s] < -eps) {
                if (r < 0
                || (number = value[r][m] / value[r][s] - value[i][m] / value[i][s]) < -eps
                || number < eps && index[r + m] > index[i + m]) {
                     r = i;
                }
            }
        }
        if (r < 0) {
            //    Solution is unbounded.
            return std::vector<double>();
        }
    }
    if (value[n + 1][m] < -eps) {
        //    No solution.
        return std::vector<double>();
    }
    std::vector<double> answer(m - 1);
    for (int i = m; i < n + m; ++i) {
        if (index[i] < m - 1) {
            answer[index[i]] = value[i - m][m];
        }
    }
    return answer;
}
```

2.4 自适应辛普森

```
double area(const double &left, const double &right) {
    double mid = (left + right) / 2;
    return (right - left) * (calc(left) + 4 * calc(mid) + calc(right)) / 6;
}

double simpson(const double &left, const double &right,
               const double &eps, const double &area_sum) {
    double mid = (left + right) / 2;
    double area_left = area(left, mid);
    double area_right = area(mid, right);
    double area_total = area_left + area_right;
    if (std::abs(area_total - area_sum) < 15 * eps) {
        return area_total + (area_total - area_sum) / 15;
    }
    return simpson(left, mid, eps / 2, area_left)
         + simpson(mid, right, eps / 2, area_right);
}

double simpson(const double &left, const double &right, const double &eps) {
    return simpson(left, right, eps, area(left, right));
}
```

2.5 牛顿迭代法

2.6 多项式方程求解

2.7 最小二乘法

3 数据结构

3.1 平衡的二叉查找树

3.1.1 Treap

```
class Node {
public:
    Node *child[2];
    int key;
    int size, priority;

    Node(Node *left, Node *right, int key) : key(key), size(1), priority(rand()) {
        child[0] = left;
```

```
        child[1] = right;
    }

    void update() {
        size = child[0]->size + 1 + child[1]->size;
    }
};

Node *null;

void rotate(Node *&x, int dir) {
    Node *y = x->child[dir];
    x->child[dir] = y->child[dir ^ 1];
    y->child[dir ^ 1] = x;
    x->update();
    y->update();
    x = y;
}

void insert(Node *&x, int key) {
    if (x == null) {
        x = new Node(null, null, key);
    } else {
        insert(x->child[key > x->key], key);
        if (x->child[key > x->key]->priority < x->priority) {
            rotate(x, key > x->key);
        }
        x->update();
    }
}

void remove(Node *&x, int key) {
    if (x->key != key) {
        remove(x->child[key > x->key], key);
    } else if (x->child[0] == null && x->child[1] == null) {
        x = null;
        return;
    } else {
        int dir = x->child[0]->priority > x->child[1]->priority;
        rotate(x, dir);
        remove(x->child[dir ^ 1], key);
    }
```

```
    x->update();
}

void build() {
    null = new Node(NULL, NULL, 0);
    null->child[0] = null->child[1] = null;
    null->size = 0;
    null->priority = RAND_MAX;
}
```

3.1.2 Splay

```
class Node {
public:
    Node *child[2], *father;
    int size;
    int key;

    Node(int key = 0);

    int side() {
        return father->child[1] == this;
    }

    void set(Node *son, int dir) {
        child[dir] = son;
        son->father = this;
    }

    void modify();

    void update() {
        size = child[0]->size + 1 + child[1]->size;
    }

    void release();
};

Node *null, *root;

Node::Node(int key) : size(1), key(key) {
    child[0] = child[1] = father = null;
}
```

```cpp
void Node::modify() {
    if (this == null) {
        return;
    }
}

void rotate(Node *x) {
    int dir = x->side();
    Node *p = x->father;
    p->release();
    x->release();
    p->set(x->child[dir ^ 1], dir);
    p->father->set(x, p->side());
    x->set(p, dir ^ 1);
    if (p == root) {
        root = x;
    }
    p->update();
    x->update();
}

void splay(Node *x, Node *target = null) {
    for (x->release(); x->father != target; ) {
        if (x->father->father == target) {
            rotate(x);
        } else {
            x->side() == x->father->side()
            ? (rotate(x->father), rotate(x))
            : (rotate(x), rotate(x));
        }
    }
    x->update();
}

Node* kth(int size) {
    Node *x = root;
    for (; x->child[0]->size + 1 != size; ) {
        x->release();
        if (x->child[0]->size + 1 > size) {
            x = x->child[0];
        } else {
```

```
                size -= x->child[0]->size + 1;
                x = x->child[1];
            }
        }
        return x;
    }

    void select(int left, int right) {
        splay(kth(right + 2));
        splay(kth(left), root);
    }

    void insert(int pos, int n, int key[]) {
        select(pos, pos - 1);
        Node *x = root->child[0];
        for (int i = 0; i < n; ++i) {
            Node *now = new Node(key[i]);
            x->set(now, 1);
            x = now;
        }
        splay(x);
    }

    void solve(int left, int right) {
        select(left, right);
        root->child[0]->child[1]->solve();
        root->child[0]->update();
        root->update();
    }

    void build() {
        null = new Node();
        null->size = 0;
        root = new Node();
        Node *blank = new Node();
        root->set(blank, 1);
        splay(blank);
    }
```

3.2 坚固的数据结构

3.2.1 坚固的线段树

```
class Node {
public:
    Node *left, *right;
    int value;

    Node(Node *left, Node *right, int value) : left(left), right(right), value(value) {}

    Node* modify(int l, int r, int ql, int qr, int delta);
    int query(int l, int r, int qx);
};

Node* null;

Node* Node::modify(int l, int r, int ql, int qr, int delta) {
    if (qr < l || r < ql) {
        return this;
    }
    if (ql <= l && r <= qr) {
        return new Node(this->left, this->right, this->value + delta);
    }
    int mid = l + r >> 1;
    return new Node(this->left->modify(l, mid, ql, qr, delta),
                    this->right->modify(mid + 1, r, ql, qr, delta),
                    this->value);
}

int Node::query(int l, int r, int qx) {
    if (qx < l || r < qx) {
        return 0;
    }
    if (qx <= l && r <= qx) {
        return this->value;
    }
    int mid = l + r >> 1;
    return this->left->query(l, mid, qx)
         + this->right->query(mid + 1, r, qx)
         + this->value;
}
```

```
void build() {
    null = new Node(NULL, NULL, 0);
    null->left = null->right = null;
}
```

3.2.2 坚固的平衡树

```
class Node {
public:
    Node *left, *right;
    int size;

    Node();
    std::pair<Node*, Node*> split(int size);

    Node* update() {
        size = left->size + 1 + right->size;
        return this;
    }
};

bool random(int a, int b) {
    return rand() % (a + b) < a;
}

Node *null;

Node::Node() : left(null), right(null), size(1) {}

Node* merge(Node *x, Node *y) {
    if (x == null) {
        return y;
    }
    if (y == null) {
        return x;
    }
    if (random(x->size, y->size)) {
        x->right = merge(x->right, y);
        return x->update();
    } else {
        y->left = merge(x, y->left);
        return y->update();
    }
```

```
}

std::pair<Node*, Node*> Node::split(int size) {
    if (this == null) {
        return std::make_pair(null, null);
    }
    if (size <= left->size) {
        std::pair<Node*, Node*> result = left->split(size);
        left = null;
        return std::make_pair(result.first, merge(result.second, this->update()));
    } else {
        std::pair<Node*, Node*> result = right->split(size - left->size - 1);
        right = null;
        return std::make_pair(merge(this->update(), result.first), result.second);
    }
}

void build() {
    null = new Node();
    null->size = 0;
}
```

3.2.3 坚固的字符串

1. ext 库中的 rope

```
#include <ext/rope>

using __gnu_cxx::crope;
using __gnu_cxx::rope;

crope a, b;

int main(void) {
    a = b.substr(pos, len);     // [pos, pos + len)
    a = b.substr(pos);          // [pos, pos]
    b.c_str();                  // might lead to memory leaks
    b.delete_c_str();           // delete the c_str that created before
    a.insert(pos, text);        // insert text before position pos
    a.erase(pos, len);          // erase [pos, pos + len)
}
```

2. 可持久化平衡树实现的 rope

```cpp
class Rope {
private:
    class Node {
    public:
        Node *left, *right;
        int size;
        char key;

        Node(char key = 0, Node *left = NULL, Node *right = NULL)
                : key(key), left(left), right(right) {
            update();
        }

        void update() {
            size = (left ? left->size : 0) + 1 + (right ? right->size : 0);
        }

        std::string to_string() {
            return (left ? left->to_string() : "") + key
                 + (right ? right->to_string() : "");
        }
    };

    bool random(int a, int b) {
        return rand() % (a + b) < a;
    }

    Node* merge(Node *x, Node *y) {
        if (!x) {
            return y;
        }
        if (!y) {
            return x;
        }
        if (random(x->size, y->size)) {
            return new Node(x->key, x->left, merge(x->right, y));
        } else {
            return new Node(y->key, merge(x, y->left), y->right);
        }
    }

    std::pair<Node*, Node*> split(Node *x, int size) {
```

```
        if (!x) {
            return std::make_pair<Node*, Node*>(NULL, NULL);
        }
        if (size == 0) {
            return std::make_pair<Node*, Node*>(NULL, x);
        }
        if (size > x->size) {
            return std::make_pair<Node*, Node*>(x, NULL);
        }
        if (x->left && size <= x->left->size) {
            std::pair<Node*, Node*> part =
                split(x->left, size);
            return std::make_pair(part.first, new Node(x->key, part.second, x->right));
        } else {
            std::pair<Node*, Node*> part =
                split(x->right, size - (x->left ? x->left->size : 0) - 1);
            return std::make_pair(new Node(x->key, x->left, part.first), part.second);
        }
    }

    Node* build(const std::string &text, int left, int right) {
        if (left > right) {
            return NULL;
        }
        int mid = left + right >> 1;
        return new Node(text[mid],
                        build(text, left, mid - 1),
                        build(text, mid + 1, right));
    }

public:
    Node *root;

    Rope() {
        root = NULL;
    }

    Rope(const std::string &text) {
        root = build(text, 0, (int)text.length() - 1);
    }

    Rope(const Rope &other) {
```

```
    root = other.root;
}

Rope& operator = (const Rope &other) {
    if (this == &other) {
        return *this;
    }
    root = other.root;
    return *this;
}

int size() {
    return root ? root->size : 0;
}

void insert(int pos, const std::string &text) {
    if (pos < 0 || pos > size()) {
        throw "Out of range";
    }
    std::pair<Node*, Node*> part = split(root, pos);
    root = merge(merge(part.first, build(text, 0, (int)text.length() - 1)),
                 part.second);
}

void erase(int left, int right) {
    if (left < 0 || left >= size() ||
        right < 1 || right > size()) {
        throw "Out of range";
    }
    if (left >= right) {
        return;
    }
    std::pair<Node*, Node*> part = split(root, left);
    root = merge(part.first, split(part.second, right - left).second);
}

std::string substr(int left, int right) {
    if (left < 0 || left >= size() ||
        right < 1 || right > size()) {
        throw "Out of range";
    }
    if (left >= right) {
```

```
            return "";
        }
        return split(split(root, left).second, right - left).first->to_string();
    }

    void copy(int left, int right, int pos) {
        if (left < 0 || left >= size() ||
            right < 1 || right > size() ||
            pos < 0 || pos > size()) {
            throw "Out of range";
        }
        if (left >= right) {
            return;
        }
        std::pair<Node*, Node*> part = split(root, pos);
        root = merge(merge(part.first,
                           split(split(root, left).second, right - left).first),
                     part.second);
    }
};
```

3.2.4 坚固的左偏树

```
class Node {
public:
    Node *left, *right;
    int key, dist;

    Node(int key) : left(NULL), right(NULL), key(key), dist(0) {}

    Node* update() {
        if (!left || (right && left->dist < right->dist)) {
            std::swap(left, right);
        }
        dist = right ? right->dist + 1 : 0;
        return this;
    }
};

Node* merge(Node *x, Node *y) {
    if (!x) {
        return y;
    }
```

```
    if (!y) {
        return x;
    }
    if (x->key < y->key) {
        x = new Node(*x);
        x->right = merge(x->right, y);
        return x->update();
    } else {
        y = new Node(*y);
        y->right = merge(x, y->right);
        return y->update();
    }
}
```

3.3 树上的魔术师

3.3.1 轻重树链剖分

```
int father[N], height[N], size[N], son[N], top[N], pos[N], data[N];

void build(int root) {
    std::vector<int> queue;
    father[root] = -1;
    height[root] = 0;
    queue.push_back(root);
    for (int head = 0; head < (int)queue.size(); ++head) {
        int x = queue[head];
        for (int i = 0; i < (int)edge[x].size(); ++i) {
            int y = edge[x][i];
            if (y != father[x]) {
                father[y] = x;
                height[y] = height[x] + 1;
                queue.push_back(y);
            }
        }
    }
    for (int index = n - 1; index >= 0; --index) {
        int x = queue[index];
        size[x] = 1;
        son[x] = -1;
        for (int i = 0; i < (int)edge[x].size(); ++i) {
            int y = edge[x][i];
            if (y != father[x]) {
```

```
                size[x] += size[y];
                if (son[x] == -1 || size[son[x]] < size[y]) {
                    son[x] = y;
                }
            }
        }
    }
    std::fill(top, top + n, 0);
    int counter = 0;
    for (int index = 0; index < n; ++index) {
        int x = queue[index];
        if (top[x] == 0) {
            for (int y = x; y != -1; y = son[y]) {
                top[y] = x;
                pos[y] = ++counter;
                data[counter] = value[y];
            }
        }
    }
    build(1, 1, n);
}

void solve(int x, int y) {
    while (true) {
        if (top[x] == top[y]) {
            if (x == y) {
                solve(1, 1, n, pos[x], pos[x]);
            } else {
                if (height[x] < height[y]) {
                    solve(1, 1, n, pos[x], pos[y]);
                } else {
                    solve(1, 1, n, pos[y], pos[x]);
                }
            }
            break;
        }
        if (height[top[x]] > height[top[y]]) {
            solve(1, 1, n, pos[top[x]], pos[x]);
            x = father[top[x]];
        } else {
            solve(1, 1, n, pos[top[y]], pos[y]);
            y = father[top[y]];
```

```
        }
    }
}
```

3.3.2 Link Cut Tree

```
class Node {
public:
    T data, sum;
    Node *father, *vfather;
    Node *child[2];
    int index;
    bool reverse;

    Node() {}

    Node(int index, const T &data) : index(index), data(data) {
        sum = data;
        father = NULL;
        vfather = NULL;
        child[0] = NULL;
        child[1] = NULL;
    }

    void update() {
        if (child[0]) {
            child[0]->release();
        }
        if (child[1]) {
            child[1]->release();
        }
        sum = (child[0] ? child[0]->sum : T()) + data + (child[1] ? child[1]->sum : T());
    }

    void release() {
        if (reverse) {
            if (child[0]) {
                child[0]->reverse ^= 1;
            }
            if (child[1]) {
                child[1]->reverse ^= 1;
            }
            std::swap(child[0], child[1]);
```

```
            reverse = false;
        }
    }
};

void rotate(Node *x, bool dir){
    Node *y = x->father;
    if (y->father) {
        y->father->child[y->father->child[1] == y] = x;
    }
    x->father = y->father;
    x->vfather = y->vfather;
    y->vfather = NULL;
    if (x->child[dir ^ 1]) {
        x->child[dir ^ 1]->father = y;
    }
    y->child[dir] = x->child[dir ^ 1];
    x->child[dir ^ 1] = y;
    y->father = x;
    y->update();
}

void splay(Node *x, Node *target = NULL){
    for (x->release(); x->father != target; ) {
        Node *y = x->father;
        if (y->father == target){
            y->release();
            x->release();
            bool dir = (y->child[1] == x);
            rotate(x, dir);
        } else {
            y->father->release();
            y->release();
            x->release();
            bool dir = (y->child[1] == x);
            if ((y->father->child[1] == y) == dir) {
                rotate(y, dir);
                rotate(x, dir);
            } else {
                rotate(x, dir);
                rotate(x, dir ^ 1);
            }
```

```
        }
    }
    x->update();
}

Node* access(Node *x){
    splay(x);
    if (x->child[0]) {
        x->release();
        x->child[0]->vfather = x;
        x->child[0]->father = NULL;
        x->child[0] = NULL;
        x->update();
    }
    Node *y = x;
    if (x->vfather) {
        y = access(x->vfather);
        x->vfather->child[0] = x;
        x->father = x->vfather;
        x->vfather->update();
    }
    splay(x);
    return y;
}

void addEdge(Node *x, Node *y) {
    access(x);
    Node *w = access(y);
    splay(x);
    if (x->vfather == w || x == w) {
        throw "Circle exists";
    }
    x->reverse ^= 1;
    access(y);
    y->child[0] = x;
    x->father = y;
    y->update();
}

void eraseEdge(Node *x, Node *y) {
    if (x == y) {
        throw "Not connected";
```

```
    }
    access(x);
    if (access(y) == x){
        splay(x, y);
        if (x->child[0]) {
            throw "Not connected";
        }
        y->release();
        y->child[1]->father = NULL;
        y->child[1]->vfather = NULL;
        y->child[1] = NULL;
        y->update();
    } else {
        splay(x);
        if (x->vfather != y || x->child[1]) {
            throw "Not connected";
        }
        access(x);
        x->release();
        x->child[1]->father = NULL;
        x->child[1]->vfather = NULL;
        x->child[1] = NULL;
        x->update();
    }
}

void modify(Node *x, const T &v) {
    access(x);
    x->data = v;
    x->update();
}

T query(Node *x, Node *y) {
    access(x);
    Node *u = access(y);
    T sum = u->data;
    splay(u);
    if (u->child[0]) {
        sum = sum + u->child[0]->sum;
    }
    access(x);
    splay(u);
```

```
    if (u->child[0]) {
        sum = sum + u->child[0]->sum;
    }
    return sum;
}

Node* lca(Node *x, Node *y) {
    access(x);
    return access(y);
}

Node* root(Node *x) {
    access(x);
    for (; ; x = x->child[1]) {
        x->release();
        if (!x->child[1]) {
            break;
        }
    }
    return x;
}

void evert(Node *x) {
    access(x);
    x->reverse ^= 1;
}

int n;
std::vector<Node> node;

void build(const std::vector<std::pair<int, int> > &edge, const std::vector<T> &weight) {
    node.clear();
    for (int i = 0; i < n; ++i) {
        node.push_back(Node(i, weight[i]));
    }
    for (int i = 0; i < (int)edge.size(); ++i) {
        int x = edge[i].first, y = edge[i].second;
        if (root(x) == root(y)) {
            throw "Circle exists";
        }
        addEdge(x, y);
    }
```

```
}
```

3.4 k-d 树

```
long long norm(const long long &x) {
    //    For manhattan distance
    return std::abs(x);
    //    For euclid distance
    return x * x;
}

struct Point {
    int x, y, id;

    const int& operator [] (int index) const {
        if (index == 0) {
            return x;
        } else {
            return y;
        }
    }

    friend long long dist(const Point &a, const Point &b) {
        long long result = 0;
        for (int i = 0; i < 2; ++i) {
            result += norm(a[i] - b[i]);
        }
        return result;
    }
} point[N];

struct Rectangle {
    int min[2], max[2];

    Rectangle() {
        min[0] = min[1] = INT_MAX;
        max[0] = max[1] = INT_MIN;
    }

    void add(const Point &p) {
        for (int i = 0; i < 2; ++i) {
            min[i] = std::min(min[i], p[i]);
            max[i] = std::max(max[i], p[i]);
```

```
        }
    }

    long long dist(const Point &p) {
        long long result = 0;
        for (int i = 0; i < 2; ++i) {
            //    For minimum distance
            result += norm(std::min(std::max(p[i], min[i]), max[i]) - p[i]);
            //    For maximum distance
            result += std::max(norm(max[i] - p[i]), norm(min[i] - p[i]));
        }
        return result;
    }
};

struct Node {
    Point seperator;
    Rectangle rectangle;
    int child[2];

    void reset(const Point &p) {
        seperator = p;
        rectangle = Rectangle();
        rectangle.add(p);
        child[0] = child[1] = 0;
    }
} tree[N << 1];

int size, pivot;

bool compare(const Point &a, const Point &b) {
    if (a[pivot] != b[pivot]) {
        return a[pivot] < b[pivot];
    }
    return a.id < b.id;
}

int build(int l, int r, int type = 1) {
    pivot = type;
    if (l >= r) {
        return 0;
    }
```

```
    int x = ++size;
    int mid = l + r >> 1;
    std::nth_element(point + l, point + mid, point + r, compare);
    tree[x].reset(point[mid]);
    for (int i = l; i < r; ++i) {
        tree[x].rectangle.add(point[i]);
    }
    tree[x].child[0] = build(l, mid, type ^ 1);
    tree[x].child[1] = build(mid + 1, r, type ^ 1);
    return x;
}

int insert(int x, const Point &p, int type = 1) {
    pivot = type;
    if (x == 0) {
        tree[++size].reset(p);
        return size;
    }
    tree[x].rectangle.add(p);
    if (compare(p, tree[x].seperator)) {
        tree[x].child[0] = insert(tree[x].child[0], p, type ^ 1);
    } else {
        tree[x].child[1] = insert(tree[x].child[1], p, type ^ 1);
    }
    return x;
}

//    For minimum distance
void query(int x, const Point &p, std::pair<long long, int> &answer, int type = 1) {
    pivot = type;
    if (x == 0 || tree[x].rectangle.dist(p) > answer.first) {
        return;
    }
    answer = std::min(answer,
             std::make_pair(dist(tree[x].seperator, p), tree[x].seperator.id));
    if (compare(p, tree[x].seperator)) {
        query(tree[x].child[0], p, answer, type ^ 1);
        query(tree[x].child[1], p, answer, type ^ 1);
    } else {
        query(tree[x].child[1], p, answer, type ^ 1);
        query(tree[x].child[0], p, answer, type ^ 1);
    }
```

```
}

std::priority_queue<std::pair<long long, int> > answer;

void query(int x, const Point &p, int k, int type = 1) {
    pivot = type;
    if (x == 0 ||
        (int)answer.size() == k && tree[x].rectangle.dist(p) > answer.top().first) {
        return;
    }
    answer.push(std::make_pair(dist(tree[x].seperator, p), tree[x].seperator.id));
    if ((int)answer.size() > k) {
        answer.pop();
    }
    if (compare(p, tree[x].seperator)) {
        query(tree[x].child[0], p, k, type ^ 1);
        query(tree[x].child[1], p, k, type ^ 1);
    } else {
        query(tree[x].child[1], p, k, type ^ 1);
        query(tree[x].child[0], p, k, type ^ 1);
    }
}
```

4 图论

4.1 强连通分量

```
int stamp, comps, top;
int dfn[N], low[N], comp[N], stack[N];

void tarjan(int x) {
    dfn[x] = low[x] = ++stamp;
    stack[top++] = x;
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        if (!dfn[y]) {
            tarjan(y);
            low[x] = std::min(low[x], low[y]);
        } else if (!comp[y]) {
            low[x] = std::min(low[x], dfn[y]);
        }
    }
```

```cpp
        if (low[x] == dfn[x]) {
            comps++;
            do {
                int y = stack[--top];
                comp[y] = comps;
            } while (stack[top] != x);
        }
    }

void solve() {
    stamp = comps = top = 0;
    std::fill(dfn, dfn + n, 0);
    std::fill(comp, comp + n, 0);
    for (int i = 0; i < n; ++i) {
        if (!dfn[i]) {
            tarjan(i);
        }
    }
}
```

4.2 双连通分量

4.2.1 点双连通分量

4.2.2 边双连通分量

4.3 2-SAT 问题

```cpp
int stamp, comps, top;
int dfn[N], low[N], comp[N], stack[N];

void add(int x, int a, int y, int b) {
    edge[x << 1 | a].push_back(y << 1 | b);
}

void tarjan(int x) {
    dfn[x] = low[x] = ++stamp;
    stack[top++] = x;
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        if (!dfn[y]) {
            tarjan(y);
            low[x] = std::min(low[x], low[y]);
        } else if (!comp[y]) {
```

```
            low[x] = std::min(low[x], dfn[y]);
        }
    }
    if (low[x] == dfn[x]) {
        comps++;
        do {
            int y = stack[--top];
            comp[y] = comps;
        } while (stack[top] != x);
    }
}

bool solve() {
    int counter = n + n + 1;
    stamp = top = comps = 0;
    std::fill(dfn, dfn + counter, 0);
    std::fill(comp, comp + counter, 0);
    for (int i = 0; i < counter; ++i) {
        if (!dfn[i]) {
            tarjan(i);
        }
    }
    for (int i = 0; i < n; ++i) {
        if (comp[i << 1] == comp[i << 1 | 1]) {
            return false;
        }
        answer[i] = (comp[i << 1 | 1] < comp[i << 1]);
    }
    return true;
}
```

4.4 二分图最大匹配

4.4.1 Hungary 算法

时间复杂度：$\mathcal{O}(V \cdot E)$

```
int n, m, stamp;
int match[N], visit[N];

bool dfs(int x) {
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        if (visit[y] != stamp) {
```

```
            visit[y] = stamp;
            if (match[y] == -1 || dfs(match[y])) {
                match[y] = x;
                return true;
            }
        }
    }
    return false;
}

int solve() {
    std::fill(match, match + m, -1);
    int answer = 0;
    for (int i = 0; i < n; ++i) {
        stamp++;
        answer += dfs(i);
    }
    return answer;
}
```

4.4.2 Hopcroft Karp 算法

时间复杂度：$\mathcal{O}(\sqrt{V} \cdot E)$

```
int matchx[N], matchy[N], level[N];

bool dfs(int x) {
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        int w = matchy[y];
        if (w == -1 || level[x] + 1 == level[w] && dfs(w)) {
            matchx[x] = y;
            matchy[y] = x;
            return true;
        }
    }
    level[x] = -1;
    return false;
}

int solve() {
    std::fill(matchx, matchx + n, -1);
    std::fill(matchy, matchy + m, -1);
```

```
    for (int answer = 0; ; ) {
        std::vector<int> queue;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1) {
                level[i] = 0;
                queue.push_back(i);
            } else {
                level[i] = -1;
            }
        }
        for (int head = 0; head < (int)queue.size(); ++head) {
            int x = queue[head];
            for (int i = 0; i < (int)edge[x].size(); ++i) {
                int y = edge[x][i];
                int w = matchy[y];
                if (w != -1 && level[w] < 0) {
                    level[w] = level[x] + 1;
                    queue.push_back(w);
                }
            }
        }
        int delta = 0;
        for (int i = 0; i < n; ++i) {
            if (matchx[i] == -1 && dfs(i)) {
                delta++;
            }
        }
        if (delta == 0) {
            return answer;
        } else {
            answer += delta;
        }
    }
}
```

4.5 二分图最大权匹配

时间复杂度：$\mathcal{O}(V^4)$

```
int labelx[N], labely[N], match[N], slack[N];
bool visitx[N], visity[N];

bool dfs(int x) {
```

```
    visitx[x] = true;
    for (int y = 0; y < n; ++y) {
        if (visity[y]) {
            continue;
        }
        int delta = labelx[x] + labely[y] - graph[x][y];
        if (delta == 0) {
            visity[y] = true;
            if (match[y] == -1 || dfs(match[y])) {
                match[y] = x;
                return true;
            }
        } else {
            slack[y] = std::min(slack[y], delta);
        }
    }
    return false;
}

int solve() {
    for (int i = 0; i < n; ++i) {
        match[i] = -1;
        labelx[i] = INT_MIN;
        labely[i] = 0;
        for (int j = 0; j < n; ++j) {
            labelx[i] = std::max(labelx[i], graph[i][j]);
        }
    }
    for (int i = 0; i < n; ++i) {
        while (true) {
            std::fill(visitx, visitx + n, 0);
            std::fill(visity, visity + n, 0);
            for (int j = 0; j < n; ++j) {
                slack[j] = INT_MAX;
            }
            if (dfs(i)) {
                break;
            }
            int delta = INT_MAX;
            for (int j = 0; j < n; ++j) {
                if (!visity[j]) {
                    delta = std::min(delta, slack[j]);
```

```
                }
            }
            for (int j = 0; j < n; ++j) {
                if (visitx[j]) {
                    labelx[j] -= delta;
                }
                if (visity[j]) {
                    labely[j] += delta;
                } else {
                    slack[j] -= delta;
                }
            }
        }
    }
    int answer = 0;
    for (int i = 0; i < n; ++i) {
        answer += graph[match[i]][i];
    }
    return answer;
}
```

4.6 最大流

时间复杂度：$\mathcal{O}(V^2 \cdot E)$

```
struct EdgeList {
    int size;
    int last[N];
    int succ[M], other[M], flow[M];
    void clear(int n) {
        size = 0;
        fill(last, last + n, -1);
    }
    void add(int x, int y, int c) {
        succ[size] = last[x];
        last[x] = size;
        other[size] = y;
        flow[size++] = c;
    }
} e;

int n, source, target;
int dist[N], curr[N];
```

```
void add(int x, int y, int c) {
    e.add(x, y, c);
    e.add(y, x, 0);
}

bool relabel() {
    std::vector<int> queue;
    for (int i = 0; i < n; ++i) {
        curr[i] = e.last[i];
        dist[i] = -1;
    }
    queue.push_back(target);
    dist[target] = 0;
    for (int head = 0; head < (int)queue.size(); ++head) {
        int x = queue[head];
        for (int i = e.last[x]; ~i; i = e.succ[i]) {
            int y = e.other[i];
            if (e.flow[i ^ 1] && dist[y] == -1) {
                dist[y] = dist[x] + 1;
                queue.push_back(y);
            }
        }
    }
    return ~dist[source];
}

int dfs(int x, int answer) {
    if (x == target) {
        return answer;
    }
    int delta = answer;
    for (int &i = curr[x]; ~i; i = e.succ[i]) {
        int y = e.other[i];
        if (e.flow[i] && dist[x] == dist[y] + 1) {
            int number = dfs(y, std::min(e.flow[i], delta));
            e.flow[i] -= number;
            e.flow[i ^ 1] += number;
            delta -= number;
        }
        if (delta == 0) {
            break;
```

```
        }
    }
    return answer - delta;
}

int solve() {
    int answer = 0;
    while (relabel()) {
        answer += dfs(source, INT_MAX));
    }
    return answer;
}
```

4.7 上下界网络流

$B(u,v)$ 表示边 (u,v) 流量的下界，$C(u,v)$ 表示边 (u,v) 流量的上界，$F(u,v)$ 表示边 (u,v) 的流量。设 $G(u,v) = F(u,v) - B(u,v)$，显然有

$$0 \le G(u,v) \le C(u,v) - B(u,v)$$

4.7.1 无源汇的上下界可行流

建立超级源点 S^* 和超级汇点 T^*，对于原图每条边 (u,v) 在新网络中连如下三条边：$S^* \to v$，容量为 $B(u,v)$；$u \to T^*$，容量为 $B(u,v)$；$u \to v$，容量为 $C(u,v) - B(u,v)$。最后求新网络的最大流，判断从超级源点 S^* 出发的边是否都满流即可，边 (u,v) 的最终解中的实际流量为 $G(u,v) + B(u,v)$。

4.7.2 有源汇的上下界可行流

从汇点 T 到源点 S 连一条上界为 ∞，下界为 0 的边。按照**无源汇的上下界可行流**一样做即可，流量即为 $T \to S$ 边上的流量。

4.7.3 有源汇的上下界最大流

1. 在**有源汇的上下界可行流**中，从汇点 T 到源点 S 的边改为连一条上界为 ∞，下届为 x 的边。x 满足二分性质，找到最大的 x 使得新网络存在**无源汇的上下界可行流**即为原图的最大流。
2. 从汇点 T 到源点 S 连一条上界为 ∞，下界为 0 的边，变成无源汇的网络。按照**无源汇的上下界可行流**的方法，建立超级源点 S^* 和超级汇点 T^*，求一遍 $S^* \to T^*$ 的最大流，再将从汇点 T 到源点 S 的这条边拆掉，求一次 $S \to T$ 的最大流即可。

4.7.4 有源汇的上下界最小流

1. 在**有源汇的上下界可行流**中，从汇点 T 到源点 S 的边改为连一条上界为 x，下界为 0 的边。x 满足二分性质，找到最小的 x 使得新网络存在**无源汇的上下界可行流**即为原图的最小流。

2. 按照无源汇的上下界可行流的方法，建立超级源点 S^* 与超级汇点 T^*，求一遍 $S^* \to T^*$ 的最大流，但是注意这一次不加上汇点 T 到源点 S 的这条边，即不使之改为无源汇的网络去求解。求完后，再加上那条汇点 T 到源点 S 上界 ∞ 的边。因为这条边下界为 0，所以 S^*，T^* 无影响，再直接求一次 $S^* \to T^*$ 的最大流。若超级源点 S^* 出发的边全部满流，则 $T \to S$ 边上的流量即为原图的最小流，否则无解。

4.8 最小费用最大流

4.8.1 稀疏图

时间复杂度：$\mathcal{O}(V \cdot E^2)$

```
struct EdgeList {
    int size;
    int last[N];
    int succ[M], other[M], flow[M], cost[M];
    void clear(int n) {
        size = 0;
        std::fill(last, last + n, -1);
    }
    void add(int x, int y, int c, int w) {
        succ[size] = last[x];
        last[x] = size;
        other[size] = y;
        flow[size] = c;
        cost[size++] = w;
    }
} e;

int n, source, target;
int prev[N];

void add(int x, int y, int c, int w) {
    e.add(x, y, c, w);
    e.add(y, x, 0, -w);
}

bool augment() {
    static int dist[N], occur[N];
    std::vector<int> queue;
    std::fill(dist, dist + n, INT_MAX);
    std::fill(occur, occur + n, 0);
    dist[source] = 0;
    occur[source] = true;
```

```
    queue.push_back(source);
    for (int head = 0; head < (int)queue.size(); ++head) {
        int x = queue[head];
        for (int i = e.last[x]; ~i; i = e.succ[i]) {
            int y = e.other[i];
            if (e.flow[i] && dist[y] > dist[x] + e.cost[i]) {
                dist[y] = dist[x] + e.cost[i];
                prev[y] = i;
                if (!occur[y]) {
                    occur[y] = true;
                    queue.push_back(y);
                }
            }
        }
        occur[x] = false;
    }
    return dist[target] < INT_MAX;
}

std::pair<int, int> solve() {
    std::pair<int, int> answer = std::make_pair(0, 0);
    while (augment()) {
        int number = INT_MAX;
        for (int i = target; i != source; i = e.other[prev[i] ^ 1]) {
            number = std::min(number, e.flow[prev[i]]);
        }
        answer.first += number;
        for (int i = target; i != source; i = e.other[prev[i] ^ 1]) {
            e.flow[prev[i]] -= number;
            e.flow[prev[i] ^ 1] += number;
            answer.second += number * e.cost[prev[i]];
        }
    }
    return answer;
}
```

4.8.2 稠密图

使用条件：费用非负

时间复杂度：$\mathcal{O}(V \cdot E^2)$

```
struct EdgeList {
    int size;
```

```
    int last[N];
    int succ[M], other[M], flow[M], cost[M];
    void clear(int n) {
        size = 0;
        std::fill(last, last + n, -1);
    }
    void add(int x, int y, int c, int w) {
        succ[size] = last[x];
        last[x] = size;
        other[size] = y;
        flow[size] = c;
        cost[size++] = w;
    }
} e;

int n, source, target, flow, cost;
int slack[N], dist[N];
bool visit[N];

void add(int x, int y, int c, int w) {
    e.add(x, y, c, w);
    e.add(y, x, 0, -w);
}

bool relabel() {
    int delta = INT_MAX;
    for (int i = 0; i < n; ++i) {
        if (!visit[i]) {
            delta = std::min(delta, slack[i]);
        }
        slack[i] = INT_MAX;
    }
    if (delta == INT_MAX) {
        return true;
    }
    for (int i = 0; i < n; ++i) {
        if (visit[i]) {
            dist[i] += delta;
        }
    }
    return false;
}
```

```cpp
int dfs(int x, int answer) {
    if (x == target) {
        flow += answer;
        cost += answer * (dist[source] - dist[target]);
        return answer;
    }
    visit[x] = true;
    int delta = answer;
    for (int i = e.last[x]; ~i; i = e.succ[i]) {
        int y = e.other[i];
        if (e.flow[i] > 0 && !visit[y]) {
            if (dist[y] + e.cost[i] == dist[x]) {
                int number = dfs(y, std::min(e.flow[i], delta));
                e.flow[i] -= number;
                e.flow[i ^ 1] += number;
                delta -= number;
                if (delta == 0) {
                    dist[x] = INT_MIN;
                    return answer;
                }
            } else {
                slack[y] = std::min(slack[y], dist[y] + e.cost[i] - dist[x]);
            }
        }
    }
    return answer - delta;
}

std::pair<int, int> solve() {
    flow = cost = 0;
    std::fill(dist, dist + n, 0);
    do {
        do {
            fill(visit, visit + n, 0);
        } while (dfs(source, INT_MAX));
    } while (!relabel());
    return std::make_pair(flow, cost);
}
```

4.9 一般图最大匹配

时间复杂度：$\mathcal{O}(V^3)$

```cpp
int match[N], belong[N], next[N], mark[N], visit[N];
std::vector<int> queue;

int find(int x) {
    if (belong[x] != x) {
        belong[x] = find(belong[x]);
    }
    return belong[x];
}

void merge(int x, int y) {
    x = find(x);
    y = find(y);
    if (x != y) {
        belong[x] = y;
    }
}

int lca(int x, int y) {
    static int stamp = 0;
    stamp++;
    while (true) {
        if (x != -1) {
            x = find(x);
            if (visit[x] == stamp) {
                return x;
            }
            visit[x] = stamp;
            if (match[x] != -1) {
                x = next[match[x]];
            } else {
                x = -1;
            }
        }
        std::swap(x, y);
    }
}

void group(int a, int p) {
    while (a != p) {
        int b = match[a], c = next[b];
        if (find(c) != p) {
```

```
            next[c] = b;
        }
        if (mark[b] == 2) {
            mark[b] = 1;
            queue.push_back(b);
        }
        if (mark[c] == 2) {
            mark[c] = 1;
            queue.push_back(c);
        }
        merge(a, b);
        merge(b, c);
        a = c;
    }
}

void augment(int source) {
    queue.clear();
    for (int i = 0; i < n; ++i) {
        next[i] = visit[i] = -1;
        belong[i] = i;
        mark[i] = 0;
    }
    mark[source] = 1;
    queue.push_back(source);
    for (int head = 0; head < (int)queue.size() && match[source] == -1; ++head) {
        int x = queue[head];
        for (int i = 0; i < (int)edge[x].size(); ++i) {
            int y = edge[x][i];
            if (match[x] == y || find(x) == find(y) || mark[y] == 2) {
                continue;
            }
            if (mark[y] == 1) {
                int r = lca(x, y);
                if (find(x) != r) {
                    next[x] = y;
                }
                if (find(y) != r) {
                    next[y] = x;
                }
                group(x, r);
                group(y, r);
```

```
            } else if (match[y] == -1) {
                next[y] = x;
                for (int u = y; u != -1; ) {
                    int v = next[u];
                    int mv = match[v];
                    match[v] = u;
                    match[u] = v;
                    u = mv;
                }
                break;
            } else {
                next[y] = x;
                mark[y] = 2;
                mark[match[y]] = 1;
                queue.push_back(match[y]);
            }
        }
    }
}

int solve() {
    std::fill(match, match + n, -1);
    for (int i = 0; i < n; ++i) {
        if (match[i] == -1) {
            augment(i);
        }
    }
    int answer = 0;
    for (int i = 0; i < n; ++i) {
        answer += (match[i] != -1);
    }
    return answer;
}
```

4.10 无向图全局最小割

时间复杂度：$\mathcal{O}(V^3)$

注意事项：处理重边时，应该对边权累加

```
int node[N], dist[N];
bool visit[N];

int solve(int n) {
```

```
    int answer = INT_MAX;
    for (int i = 0; i < n; ++i) {
        node[i] = i;
    }
    while (n > 1) {
        int max = 1;
        for (int i = 0; i < n; ++i) {
            dist[node[i]] = graph[node[0]][node[i]];
            if (dist[node[i]] > dist[node[max]]) {
                max = i;
            }
        }
        int prev = 0;
        memset(visit, 0, sizeof(visit));
        visit[node[0]] = true;
        for (int i = 1; i < n; ++i) {
            if (i == n - 1) {
                answer = std::min(answer, dist[node[max]]);
                for (int k = 0; k < n; ++k) {
                    graph[node[k]][node[prev]] =
                        (graph[node[prev]][node[k]] += graph[node[k]][node[max]]);
                }
                node[max] = node[--n];
            }
            visit[node[max]] = true;
            prev = max;
            max = -1;
            for (int j = 1; j < n; ++j) {
                if (!visit[node[j]]) {
                    dist[node[j]] += graph[node[prev]][node[j]];
                    if (max == -1 || dist[node[max]] < dist[node[j]]) {
                        max = j;
                    }
                }
            }
        }
    }
    return answer;
}
```

4.11 最小树形图

4.12 有根树的同构

时间复杂度：$\mathcal{O}(VlogV)$

```
const unsigned long long MAGIC = 4423;

unsigned long long magic[N];
std::pair<unsigned long long, int> hash[N];

void solve(int root) {
    magic[0] = 1;
    for (int i = 1; i <= n; ++i) {
        magic[i] = magic[i - 1] * MAGIC;
    }
    std::vector<int> queue;
    queue.push_back(root);
    for (int head = 0; head < (int)queue.size(); ++head) {
        int x = queue[head];
        for (int i = 0; i < (int)son[x].size(); ++i) {
            int y = son[x][i];
            queue.push_back(y);
        }
    }
    for (int index = n - 1; index >= 0; --index) {
        int x = queue[index];
        hash[x] = std::make_pair(0, 0);

        std::vector<std::pair<unsigned long long, int> > value;
        for (int i = 0; i < (int)son[x].size(); ++i) {
            int y = son[x][i];
            value.push_back(hash[y]);
        }
        std::sort(value.begin(), value.end());

        hash[x].first = hash[x].first * magic[1] + 37;
        hash[x].second++;
        for (int i = 0; i < (int)value.size(); ++i) {
            hash[x].first = hash[x].first * magic[value[i].second] + value[i].first;
            hash[x].second += value[i].second;
        }
        hash[x].first = hash[x].first * magic[1] + 41;
        hash[x].second++;
```

```
    }
}
```

4.13 度限制生成树

4.14 弦图相关

4.14.1 弦图的判定

4.14.2 弦图的团数

4.15 哈密尔顿回路（ORE 性质的图）

ORE 性质：

$$\forall x, y \in V \wedge (x, y) \notin E \quad s.t. \quad deg_x + deg_y \geq n$$

返回结果：从顶点 1 出发的一个哈密尔顿回路
使用条件：$n \geq 3$

```
int left[N], right[N], next[N], last[N];

void cover(int x) {
    left[right[x]] = left[x];
    right[left[x]] = right[x];
}

int adjacent(int x) {
    for (int i = right[0]; i <= n; i = right[i]) {
        if (graph[x][i]) {
            return i;
        }
    }
    return 0;
}

std::vector<int> solve() {
    for (int i = 1; i <= n; ++i) {
        left[i] = i - 1;
        right[i] = i + 1;
    }
    int head, tail;
    for (int i = 2; i <= n; ++i) {
        if (graph[1][i]) {
            head = 1;
            tail = i;
```

```
            cover(head);
            cover(tail);
            next[head] = tail;
            break;
        }
    }
    while (true) {
        int x;
        while (x = adjacent(head)) {
            next[x] = head;
            head = x;
            cover(head);
        }
        while (x = adjacent(tail)) {
            next[tail] = x;
            tail = x;
            cover(tail);
        }
        if (!graph[head][tail]) {
            for (int i = head, j; i != tail; i = next[i]) {
                if (graph[head][next[i]] && graph[tail][i]) {
                    for (j = head; j != i; j = next[j]) {
                        last[next[j]] = j;
                    }
                    j = next[head];
                    next[head] = next[i];
                    next[tail] = i;
                    tail = j;
                    for (j = i; j != head; j = last[j]) {
                        next[j] = last[j];
                    }
                    break;
                }
            }
        }
        next[tail] = head;
        if (right[0] > n) {
            break;
        }
        for (int i = head; i != tail; i = next[i]) {
            if (adjacent(i)) {
                head = next[i];
```

```
                tail = i;
                next[tail] = 0;
                break;
            }
        }
    }
    std::vector<int> answer;
    for (int i = head; ; i = next[i]) {
        if (i == 1) {
            answer.push_back(i);
            for (int j = next[i]; j != i; j = next[j]) {
                answer.push_back(j);
            }
            answer.push_back(i);
            break;
        }
        if (i == tail) {
            break;
        }
    }
    return answer;
}
```

5 字符串

5.1 模式匹配

5.1.1 KMP 算法

```
void build(char *pattern) {
    int length = (int)strlen(pattern + 1);
    fail[0] = -1;
    for (int i = 1, j; i <= length; ++i) {
        for (j = fail[i - 1]; j != -1 && pattern[i] != pattern[j + 1]; j = fail[j]);
        fail[i] = j + 1;
    }
}

void solve(char *text, char *pattern) {
    int length = (int)strlen(text + 1);
    for (int i = 1, j; i <= length; ++i) {
        for (j = match[i - 1]; j != -1 && text[i] != pattern[j + 1]; j = fail[j]);
        match[i] = j + 1;
```

```
    }
}
```

5.1.2 扩展 KMP 算法

返回结果：

$$next_i = lcp(text, text_{i...n-1})$$

```
void solve(char *text, int length, int *next) {
    int j = 0, k = 1;
    for (; j + 1 < length && text[j] == text[j + 1]; j++);
    next[0] = length - 1;
    next[1] = j;
    for (int i = 2; i < length; ++i) {
        int far = k + next[k] - 1;
        if (next[i - k] < far - i + 1) {
            next[i] = next[i - k];
        } else {
            j = std::max(far - i + 1, 0);
            for (; i + j < length && text[j] == text[i + j]; j++);
            next[i] = j;
            k = i;
        }
    }
}
```

5.1.3 AC 自动机

```
class Node {
public:
    Node *child[256], *fail;
    int counter;

    Node() : fail(NULL), counter(0) {
        memset(child, NULL, sizeof(child));
    }
};

void insert(Node *x, char *text) {
    int length = (int)strlen(text);
    for (int i = 0; i < length; ++i) {
        int token = (int)text[i];
        if (!x->child[token]) {
```

```
            x->child[token] = new Node();
        }
        x = x->child[token];
    }
    x->counter++;
}

void build() {
    std::vector<Node*> queue;
    queue.push_back(root->fail = root);
    for (int head = 0; head < (int)queue.size(); ++head) {
        Node *x = queue[head];
        for (int token = 0; token < 256; ++token) {
            if (x->child[token]) {
                x->child[token]->fail = (x == root) ? root : x->fail->child[token];
                x->child[token]->counter += x->child[token]->fail->counter;
                queue.push_back(x->child[token]);
            } else {
                x->child[token] = (x == root) ? root : x->fail->child[token];
            }
        }
    }
}
```

5.2 后缀三姐妹

5.2.1 后缀数组

```
int array[N], rank[N], height[N];
int counter[N], new_array[N], new_rank[N][2];
int log2[N], value[N][20];

void build(char *text, int n) {
    memset(counter, 0, sizeof(counter));
    for (int i = 0; i < n; ++i) {
        counter[(int)text[i]]++;
    }
    for (int i = 0; i < 256; ++i) {
        counter[i + 1] += counter[i];
    }
    for (int i = 0; i < n; ++i) {
        rank[i] = counter[(int)text[i]] - 1;
    }
```

```
for (int length = 1; length < n; length <<= 1) {
    for (int i = 0; i < n; ++i) {
        new_rank[i][0] = rank[i];
        new_rank[i][1] = i + length < n ? rank[i + length] + 1 : 0;
    }
    memset(counter, 0, sizeof(counter));
    for (int i = 0; i < n; ++i) {
        counter[new_rank[i][1]]++;
    }
    for (int i = 0; i < n; ++i) {
        counter[i + 1] += counter[i];
    }
    for (int i = n - 1; i >= 0; --i) {
        new_array[--counter[new_rank[i][1]]] = i;
    }
    memset(counter, 0, sizeof(counter));
    for (int i = 0; i < n; ++i) {
        counter[new_rank[i][0]]++;
    }
    for (int i = 0; i < n; ++i) {
        counter[i + 1] += counter[i];
    }
    for (int i = n - 1; i >= 0; --i) {
        array[--counter[new_rank[new_array[i]][0]]] = new_array[i];
    }
    rank[array[0]] = 0;
    for (int i = 0; i + 1 < n; ++i) {
        rank[array[i + 1]] = rank[array[i]] +
            (new_rank[array[i]][0] != new_rank[array[i + 1]][0]
          || new_rank[array[i]][1] != new_rank[array[i + 1]][1]);
    }
}
for (int i = 0, length = 0; i < n; ++i) {
    if (rank[i]) {
        int j = array[rank[i] - 1];
        while (i + length < n && j + length < n
                && text[i + length] == text[j + length]) {
            length++;
        }
        height[rank[i]] = length;
        if (length) {
            length--;
```

```
            }
        }
    }
    for (int i = 2; i <= n; ++i) {
        log2[i] = log2[i >> 1] + 1;
    }
    for (int i = 1; i < n; ++i) {
        value[i][0] = height[i];
    }
    for (int step = 1; (1 << step) <= n; ++step) {
        for (int i = 1; i + (1 << step) <= n; ++i) {
            value[i][step] = std::min(value[i][step - 1],
                                      value[i + (1 << step - 1)][step - 1]);
        }
    }
}

int lcp(int left, int right) {
    if (left > right) {
        std::swap(left, right);
    }
    int step = log2[right - left];
    return std::min(value[left + 1][step], value[right - (1 << step) + 1][step]);
}
```

5.2.2 后缀自动机

```
class Node {
public:
    Node *child[256], *parent;
    int length;

    Node(int length = 0) : parent(NULL), length(length) {
        memset(child, NULL, sizeof(child));
    }

    Node* extend(Node *start, int token) {
        Node *p = this;
        Node *np = new Node(length + 1);
        for (; p && !p->child[token]; p = p->parent) {
            p->child[token] = np;
        }
        if (!p) {
```

```
                np->parent = start;
            } else {
                Node *q = p->child[token];
                if (p->length + 1 == q->length) {
                    np->parent = q;
                } else {
                    Node *nq = new Node(p->length + 1);
                    memcpy(nq->child, q->child, sizeof(q->child));
                    nq->parent = q->parent;
                    np->parent = q->parent = nq;
                    for (; p && p->child[token] == q; p = p->parent) {
                        p->child[token] = nq;
                    }
                }
            }
            return np;
        }
};
```

5.3 回文三兄弟

5.3.1 Manacher 算法

```
void manacher(char *text, int length) {
    palindrome[0] = 1;
    for (int i = 1, j = 0; i < length; ++i) {
        if (j + palindrome[j] <= i) {
            palindrome[i] = 0;
        } else {
            palindrome[i] = std::min(palindrome[(j << 1) - i], j + palindrome[j] - i);
        }
        while (i - palindrome[i] >= 0 && i + palindrome[i] < length
                && text[i - palindrome[i]] == text[i + palindrome[i]]) {
            palindrome[i]++;
        }
        if (i + palindrome[i] > j + palindrome[j]) {
            j = i;
        }
    }
}
```

5.3.2 回文树

```
class Node {
public:
    Node *child[256], *fail;
    int length;

    Node(int length) : fail(NULL), length(length) {
        memset(child, NULL, sizeof(child));
    }
};

int size;
int text[N];
Node *odd, *even;

Node* match(Node *now) {
    for (; text[size - now->length - 1] != text[size]; now = now->fail);
    return now;
}

bool extend(Node *&last, int token) {
    text[++size] = token;
    Node *now = last;
    now = match(now);
    if (now->child[token]) {
        last = now->child[token];
        return false;
    }
    last = now->child[token] = new Node(now->length + 2);
    if (now == odd) {
        last->fail = even;
    } else {
        now = match(now->fail);
        last->fail = now->child[token];
    }
    return true;
}

void build() {
    text[size = 0] = -1;
    even = new Node(0), odd = new Node(-1);
    even->fail = odd;
```

```
}
```

5.4 循环串最小表示

```
int solve(char *text, int length) {
    int i = 0, j = 1, delta = 0;
    while (i < length && j < length && delta < length) {
        char tokeni = text[(i + delta) % length];
        char tokenj = text[(j + delta) % length];
        if (tokeni == tokenj) {
            delta++;
        } else {
            if (tokeni > tokenj) {
                i += delta + 1;
            } else {
                j += delta + 1;
            }
            if (i == j) {
                j++;
            }
            delta = 0;
        }
    }
    return std::min(i, j);
}
```

6 计算几何

6.1 二维基础

6.1.1 点类

6.1.2 凸包

```
std::vector<Point> convex_hull(std::vector<Point> point) {
    std::sort(point.begin(), point.end());
    std::vector<Point> convex;
    {
        std::vector<Point> stack;
        for (int i = 0; i < (int)point.size(); ++i) {
            while ((int)stack.size() >= 2 &&
                    sgn(det(stack[(int)stack.size() - 2], stack.back(), point[i])) <= 0) {
                stack.pop_back();
```

```
            }
            stack.push_back(point[i]);
        }
        for (int i = 0; i < (int)stack.size(); ++i) {
            convex.push_back(stack[i]);
        }
    }
    {
        std::vector<Point> stack;
        for (int i = (int)point.size() - 1; i >= 0; --i) {
            while ((int)stack.size() >= 2 &&
                    sgn(det(stack[(int)stack.size() - 2], stack.back(), point[i])) <= 0) {
                stack.pop_back();
            }
            stack.push_back(point[i]);
        }
        for (int i = 1; i < (int)stack.size() - 1; ++i) {
            convex.push_back(stack[i]);
        }
    }
    return convex;
}
```

6.1.3 半平面交

6.1.4 最近点对

```
bool comparex(const Point &a, const Point &b) {
    return sgn(a.x - b.x) < 0;
}

bool comparey(const Point &a, const Point &b) {
    return sgn(a.y - b.y) < 0;
}

double solve(const std::vector<Point> &point, int left, int right) {
    if (left == right) {
        return INF;
    }
    if (left + 1 == right) {
        return dist(point[left], point[right]);
    }
    int mid = left + right >> 1;
```

```
    double result = std::min(solve(left, mid), solve(mid + 1, right));
    std::vector<Point> candidate;
    for (int i = left; i <= right; ++i) {
        if (std::abs(point[i].x - point[mid].x) <= result) {
            candidate.push_back(point[i]);
        }
    }
    std::sort(candidate.begin(), candidate.end(), comparey);
    for (int i = 0; i < (int)candidate.size(); ++i) {
        for (int j = i + 1; j < (int)candidate.size(); ++j) {
            if (std::abs(candidate[i].y - candidate[j].y) >= result) {
                break;
            }
            result = std::min(result, dist(candidate[i], candidate[j]));
        }
    }
    return result;
}

double solve(std::vector<Point> point) {
    std::sort(point.begin(), point.end(), comparex);
    return solve(point, 0, (int)point.size() - 1);
}
```

6.2 三维基础

6.2.1 点类

6.2.2 凸包

6.2.3 绕轴旋转

6.3 多边形

6.3.1 判断点在多边形内部

```
bool point_on_line(const Point &p, const Point &a, const Point &b) {
    return sgn(det(p, a, b)) == 0 && sgn(dot(p, a, b)) <= 0;
}

bool point_in_polygon(const Point &p, const std::vector<Point> &polygon) {
    int counter = 0;
    for (int i = 0; i < (int)polygon.size(); ++i) {
        Point a = polygon[i], b = polygon[(i + 1) % (int)polygon.size()];
        if (point_on_line(p, a, b)) {
```

```
            //    Point on the boundary are excluded.
            return false;
        }
        int x = sgn(det(a, p, b));
        int y = sgn(a.y - p.y);
        int z = sgn(b.y - p.y);
        counter += (x > 0 && y <= 0 && z > 0);
        counter -= (x < 0 && z <= 0 && y > 0);
    }
    return counter;
}
```

6.3.2 旋转卡壳

6.3.3 动态凸包

6.3.4 点到凸包的切线

6.3.5 直线与凸包的交点

6.3.6 凸多边形内的最大圆

6.4 圆

6.4.1 圆类

6.4.2 圆的交集

6.4.3 最小覆盖圆

6.4.4 最小覆盖球

6.4.5 判断圆存在交集

6.4.6 圆与多边形的交集

6.5 三角形

6.5.1 三角形的内心

6.5.2 三角形的外心

6.5.3 三角形的垂心

7 其他

7.1 某年某月某日是星期几

```
int solve(int year, int month, int day) {
    int answer;
    if (month == 1 || month == 2) {
        month += 12;
        year--;
    }
    if ((year < 1752) || (year == 1752 && month < 9) ||
        (year == 1752 && month == 9 && day < 3)) {
        answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4 + 5) % 7;
    } else {
        answer = (day + 2 * month + 3 * (month + 1) / 5 + year + year / 4
               - year / 100 + year / 400) % 7;
    }
```

```
    return answer;
}
```

7.2 枚举大小为 k 的子集

使用条件：$k > 0$

```
void solve(int n, int k) {
    for (int comb = (1 << k) - 1; comb < (1 << n); ) {
        // ...
        int x = comb & -comb, y = comb + x;
        comb = (((comb & ~y) / x) >> 1) | y;
    }
}
```

7.3 环状最长公共子串

```
int n, a[N << 1], b[N << 1];

bool has(int i, int j) {
    return a[(i - 1) % n] == b[(j - 1) % n];
}

const int DELTA[3][2] = {{0, -1}, {-1, -1}, {-1, 0}};

int from[N][N];

int solve() {
    memset(from, 0, sizeof(from));
    int ret = 0;
    for (int i = 1; i <= 2 * n; ++i) {
        from[i][0] = 2;
        int left = 0, up = 0;
        for (int j = 1; j <= n; ++j) {
            int upleft = up + 1 + !!from[i - 1][j];
            if (!has(i, j)) {
                upleft = INT_MIN;
            }
            int max = std::max(left, std::max(upleft, up));
            if (left == max) {
                from[i][j] = 0;
            } else if (upleft == max) {
                from[i][j] = 1;
```

```
            } else {
                from[i][j] = 2;
            }
            left = max;
        }
        if (i >= n) {
            int count = 0;
            for (int x = i, y = n; y; ) {
                int t = from[x][y];
                count += t == 1;
                x += DELTA[t][0];
                y += DELTA[t][1];
            }
            ret = std::max(ret, count);
            int x = i - n + 1;
            from[x][0] = 0;
            int y = 0;
            while (y <= n && from[x][y] == 0) {
                y++;
            }
            for (; x <= i; ++x) {
                from[x][y] = 0;
                if (x == i) {
                    break;
                }
                for (; y <= n; ++y) {
                    if (from[x + 1][y] == 2) {
                        break;
                    }
                    if (y + 1 <= n && from[x + 1][y + 1] == 1) {
                        y++;
                        break;
                    }
                }
            }
        }
    }
    return ret;
}
```

7.4 搜索

7.4.1 Dancing Links X

8 Java

8.1 基础模板

```java
import java.io.*;
import java.util.*;
import java.math.*;

public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        Task solver = new Task();
        solver.solve(0, in, out);
        out.close();
    }
}

class Task {
    public void solve(int testNumber, InputReader in, PrintWriter out) {

    }
}

class InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;

    public InputReader(InputStream stream) {
        reader = new BufferedReader(new InputStreamReader(stream), 32768);
        tokenizer = null;
    }

    public String next() {
        while (tokenizer == null || !tokenizer.hasMoreTokens()) {
            try {
                tokenizer = new StringTokenizer(reader.readLine());
```

```java
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        return tokenizer.nextToken();
    }

    public int nextInt() {
        return Integer.parseInt(next());
    }

    public long nextLong() {
        return Long.parseLong(next());
    }
}
```

9 数学

9.1 常用数学公式

9.1.1 求和公式

1. $\sum_{k=1}^{n}(2k-1)^2 = \frac{n(4n^2-1)}{3}$
2. $\sum_{k=1}^{n} k^3 = [\frac{n(n+1)}{2}]^2$
3. $\sum_{k=1}^{n}(2k-1)^3 = n^2(2n^2-1)$
4. $\sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
5. $\sum_{k=1}^{n} k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$
6. $\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3}$
7. $\sum_{k=1}^{n} k(k+1)(k+2) = \frac{n(n+1)(n+2)(n+3)}{4}$
8. $\sum_{k=1}^{n} k(k+1)(k+2)(k+3) = \frac{n(n+1)(n+2)(n+3)(n+4)}{5}$

9.1.2 斐波那契数列

1. $fib_0 = 0, fib_1 = 1, fib_n = fib_{n-1} + fib_{n-2}$
2. $fib_{n+2} \cdot fib_n - fib_{n+1}^2 = (-1)^{n+1}$
3. $fib_{-n} = (-1)^{n-1} fib_n$
4. $fib_{n+k} = fib_k \cdot fib_{n+1} + fib_{k-1} \cdot fib_n$
5. $gcd(fib_m, fib_n) = fib_{gcd(m,n)}$

6. $fib_m | fib_n^2 \Leftrightarrow nfib_n | m$

9.1.3 错排公式

1. $D_n = (n-1)(D_{n-2} - D_{n-1})$
2. $D_n = n! \cdot (1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots + \frac{(-1)^n}{n!})$

9.1.4 莫比乌斯函数

$$\mu(n) = \begin{cases} 1 & \text{若} n = 1 \\ (-1)^k & \text{若} n \text{无平方数因子，且} n = p_1 p_2 \ldots p_k \\ 0 & \text{若} n \text{有大于1的平方数因数} \end{cases}$$

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{若} n = 1 \\ 0 & \text{其他情况} \end{cases}$$

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

$$g(x) = \sum_{n=1}^{[x]} f(\frac{x}{n}) \Leftrightarrow f(x) = \sum_{n=1}^{[x]} \mu(n) g(\frac{x}{n})$$

9.1.5 伯恩赛德引理

设 G 是一个有限群，作用在集合 X 上。对每个 g 属于 G，令 X^g 表示 X 中在 g 作用下的不动元素，轨道数（记作 $|X/G|$）由如下公式给出：

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

9.1.6 五边形数定理

设 $p(n)$ 是 n 的拆分数，有

$$p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k-1} p\left(n - \frac{k(3k-1)}{2}\right)$$

9.1.7 树的计数

1. 有根树计数：$n+1$ 个结点的有根树的个数为

$$a_{n+1} = \frac{\sum_{j=1}^{n} j \cdot a_j \cdot S_{n,j}}{n}$$

其中，

$$S_{n,j}=\sum_{i=1}^{n/j}a_{n+1-ij}=S_{n-j,j}+a_{n+1-j}$$

2. 无根树计数：当 n 为奇数时，n 个结点的无根树的个数为

$$a_n-\sum_{i=1}^{n/2}a_ia_{n-i}$$

当 n 为偶数时，n 个结点的无根树的个数为

$$a_n-\sum_{i=1}^{n/2}a_ia_{n-i}+\frac{1}{2}a_{\frac{n}{2}}(a_{\frac{n}{2}}+1)$$

3. n 个结点的完全图的生成树个数为

$$n^{n-2}$$

4. 矩阵－树定理：图 G 由 n 个结点构成，设 $\boldsymbol{A}[G]$ 为图 G 的邻接矩阵、$\boldsymbol{D}[G]$ 为图 G 的度数矩阵，则图 G 的不同生成树的个数为 $\boldsymbol{C}[G]=\boldsymbol{D}[G]-\boldsymbol{A}[G]$ 的任意一个 $n-1$ 阶主子式的行列式值。

9.1.8 欧拉公式

平面图的顶点个数、边数和面的个数有如下关系：

$$V-E+F=C+1$$

其中，V 是顶点的数目，E 是边的数目，F 是面的数目，C 是组成图形的连通部分的数目。当图是单连通图的时候，公式简化为：

$$V-E+F=2$$

9.1.9 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，其面积 A 和内部格点数目 i、边上格点数目 b 的关系：

$$A=i+\frac{b}{2}-1$$

9.1.10 牛顿恒等式

设

$$\prod_{i=1}^{n}(x-x_i)=a_n+a_{n-1}x+\cdots+a_1x^{n-1}+a_0x^n$$

$$p_k=\sum_{i=1}^{n}x_i^k$$

则

$$a_0p_k + a_1p_{k-1} + \cdots + a_{k-1}p_1 + ka_k = 0$$

特别地，对于

$$|\boldsymbol{A} - \lambda\boldsymbol{E}| = (-1)^n(a_n + a_{n-1}\lambda + \cdots + a_1\lambda^{n-1} + a_0\lambda^n)$$

有

$$p_k = Tr(\boldsymbol{A}^k)$$

9.2 平面几何公式

9.2.1 三角形

1. 半周长

$$p = \frac{a+b+c}{2}$$

2. 面积

$$S = \frac{a\cdot H_a}{2} = \frac{ab\cdot sinC}{2} = \sqrt{p(p-a)(p-b)(p-c)}$$

3. 中线

$$M_a = \frac{\sqrt{2(b^2+c^2)-a^2}}{2} = \frac{\sqrt{b^2+c^2+2bc\cdot cosA}}{2}$$

4. 角平分线

$$T_a = \frac{\sqrt{bc\cdot[(b+c)^2-a^2]}}{b+c} = \frac{2bc}{b+c}cos\frac{A}{2}$$

5. 高线

$$H_a = bsinC = csinB = \sqrt{b^2 - (\frac{a^2+b^2-c^2}{2a})^2}$$

6. 内切圆半径

$$\begin{aligned} r = \frac{S}{p} = \frac{arcsin\frac{B}{2}\cdot sin\frac{C}{2}}{sin\frac{B+C}{2}} &= 4R\cdot sin\frac{A}{2}sin\frac{B}{2}sin\frac{C}{2} \\ = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}} &= p\cdot tan\frac{A}{2}tan\frac{B}{2}tan\frac{C}{2} \end{aligned}$$

7. 外接圆半径

$$R = \frac{abc}{4S} = \frac{a}{2sinA} = \frac{b}{2sinB} = \frac{c}{2sinC}$$

9.2.2 四边形

D_1, D_2 为对角线，M 对角线中点连线，A 为对角线夹角，p 为半周长

1. $a^2+b^2+c^2+d^2 = D_1^2+D_2^2+4M^2$
2. $S = \frac{1}{2}D_1D_2sinA$

3. 对于圆内接四边形

$$ac + bd = D_1 D_2$$

4. 对于圆内接四边形

$$S = \sqrt{(p-a)(p-b)(p-c)(p-d)}$$

9.2.3 正 n 边形

R 为外接圆半径，r 为内切圆半径

1. 中心角

$$A = \frac{2\pi}{n}$$

2. 内角

$$C = \frac{n-2}{n}\pi$$

3. 边长

$$a = 2\sqrt{R^2 - r^2} = 2R \cdot sin\frac{A}{2} = 2r \cdot tan\frac{A}{2}$$

4. 面积

$$S = \frac{nar}{2} = nr^2 \cdot tan\frac{A}{2} = \frac{nR^2}{2} \cdot sinA = \frac{na^2}{4 \cdot tan\frac{A}{2}}$$

9.2.4 圆

1. 弧长

$$l = rA$$

2. 弦长

$$a = 2\sqrt{2hr - h^2} = 2r \cdot sin\frac{A}{2}$$

3. 弓形高

$$h = r - \sqrt{r^2 - \frac{a^2}{4}} = r(1 - cos\frac{A}{2}) = \frac{1}{2} \cdot arctan\frac{A}{4}$$

4. 扇形面积

$$S_1 = \frac{rl}{2} = \frac{r^2 A}{2}$$

5. 弓形面积

$$S_2 = \frac{rl - a(r-h)}{2} = \frac{r^2}{2}(A - sinA)$$

9.2.5 棱柱

1. 体积

$$V = Ah$$

A 为底面积，h 为高

2. 侧面积

$$S = lp$$

l 为棱长，p 为直截面周长

3. 全面积

$$T = S + 2A$$

9.2.6 棱锥

1. 体积

$$V = Ah$$

A 为底面积，h 为高

2. 正棱锥侧面积

$$S = lp$$

l 为棱长，p 为直截面周长

3. 正棱锥全面积

$$T = S + 2A$$

9.2.7 棱台

1. 体积

$$V = (A_1 + A_2 + \sqrt{A_1 A_2}) \cdot \frac{h}{3}$$

A_1, A_2 为上下底面积，h 为高

2. 正棱台侧面积

$$S = \frac{p_1 + p_2}{2} l$$

p_1, p_2 为上下底面周长，l 为斜高

3. 正棱台全面积

$$T = S + A_1 + A_2$$

9.2.8 圆柱

1. 侧面积

$$S = 2\pi rh$$

2. 全面积

$$T = 2\pi r(h + r)$$

3. 体积

$$V = \pi r^2 h$$

9.2.9 圆锥

1. 母线

$$l = \sqrt{h^2 + r^2}$$

2. 侧面积

$$S = \pi r l$$

3. 全面积

$$T = \pi r(l + r)$$

4. 体积

$$V = \frac{\pi}{3} r^2 h$$

9.2.10 圆台

1. 母线

$$l = \sqrt{h^2 + (r_1 - r_2)^2}$$

2. 侧面积

$$S = \pi(r_1 + r_2)l$$

3. 全面积

$$T = \pi r_1(l + r_1) + \pi r_2(l + r_2)$$

4. 体积

$$V = \frac{\pi}{3}(r_1^2 + r_2^2 + r_1 r_2)h$$

9.2.11 球

1. 全面积

$$T = 4\pi r^2$$

2. 体积

$$V = \frac{4}{3}\pi r^3$$

9.2.12 球台

1. 侧面积

$$S = 2\pi r h$$

2. 全面积

$$T = \pi(2rh + r_1^2 + r_2^2)$$

3. 体积

$$V = \frac{\pi h[3(r_1^2 + r_2^2) + h^2]}{6}$$

9.2.13 球扇形

1. 全面积

$$T = \pi r(2h + r_0)$$

   h 为球冠高，r_0 为球冠底面半径

2. 体积

$$V = \frac{2}{3}\pi r^2 h$$

9.3 立体几何公式

9.3.1 球面三角公式

设 a, b, c 是边长，A, B, C 是所对的二面角，有余弦定理

$$cosa = cosb \cdot cosc + sinb \cdot sinc \cdot cosA$$

正弦定理

$$\frac{sinA}{sina} = \frac{sinB}{sinb} = \frac{sinC}{sinc}$$

三角形面积是 $A + B + C - \pi$

9.3.2 四面体体积公式

U, V, W, u, v, w 是四面体的 6 条棱，U, V, W 构成三角形，$(U, u), (V, v), (W, w)$ 互为对棱，则

$$V = \frac{\sqrt{(s-2a)(s-2b)(s-2c)(s-2d)}}{192uvw}$$

其中

$$\begin{cases} a &=& \sqrt{xYZ}, \\ b &=& \sqrt{yZX}, \\ c &=& \sqrt{zXY}, \\ d &=& \sqrt{xyz}, \\ s &=& a+b+c+d, \\ X &=& (w-U+v)(U+v+w), \\ x &=& (U-v+w)(v-w+U), \\ Y &=& (u-V+w)(V+w+u), \\ y &=& (V-w+u)(w-u+V), \\ Z &=& (v-W+u)(W+u+v), \\ z &=& (W-u+v)(u-v+W) \end{cases}$$