

Shoka

For Manual/Intelligence

August 6, 2018

# 代码库

上海交通大学

October 24, 2013

## Contents

|          |                             |           |
|----------|-----------------------------|-----------|
| <b>1</b> | <b>计算几何</b>                 | <b>2</b>  |
| 1.1      | 半平面交 . . . . .              | 2         |
| 1.1.1    | $O(N^2)$ . . . . .          | 2         |
| 1.1.2    | $O(N \log N)$ . . . . .     | 2         |
| <b>2</b> | <b>数据结构</b>                 | <b>2</b>  |
| 2.1      | kd 树 . . . . .              | 2         |
| 2.2      | 坚固的数据结构 . . . . .           | 4         |
| 2.2.1    | 坚固的线段树 . . . . .            | 4         |
| 2.2.2    | 坚固的平衡树 . . . . .            | 5         |
| 2.3      | 后缀三姐妹 . . . . .             | 6         |
| 2.3.1    | 后缀数组 . . . . .              | 6         |
| 2.3.2    | 后缀自动机 . . . . .             | 7         |
| 2.4      | 最长回文串 Manacher 算法 . . . . . | 7         |
| <b>3</b> | <b>图论</b>                   | <b>8</b>  |
| <b>4</b> | <b>数论</b>                   | <b>8</b>  |
| 4.1      | Miller-rabin . . . . .      | 8         |
| 4.2      | Polar Rho . . . . .         | 9         |
| 4.3      | 快速傅里叶变换 . . . . .           | 9         |
| 4.4      | 直线下格点统计 . . . . .           | 10        |
| <b>5</b> | <b>Miscellaneous</b>        | <b>10</b> |
| 5.1      | 环状最长公共子串 . . . . .          | 10        |
| 5.2      | 二次剩余 . . . . .              | 11        |
| 5.3      | 五边形数定理 . . . . .            | 12        |
| 5.4      | 球面三角公式 . . . . .            | 12        |
| 5.5      | 四面体体积公式 . . . . .           | 12        |
| 5.6      | 牛顿恒等式 . . . . .             | 12        |

# 1 计算几何

## 1.1 半平面交

### 1.1.1 $O(N^2)$

```
void rebuild(const Point &a, const Point &b) {
    points[n] = points[0];
    int m = 0;
    for (int i = 0; i < n; ++ i) {
        double s_1 = det(b - a, points[i] - a);
        double s_2 = det(b - a, points[i + 1] - a);
        if (signum(s_1) * signum(s_2) < 0) {
            newPoints[m ++] = (points[i + 1] * s_2 - points[i] * s_1) / (s_2 - s_1);
        }
        if (signum(det(b - a, points[i + 1] - a)) >= 0) {
            newPoints[m ++] = points[i + 1];
        }
    }
    n = m;
    copy(newPoints, newPoints + n, points);
}
```

### 1.1.2 $O(N \log N)$

```
bool check(const Plane &u, const Plane &v, const Plane &w) {
    return intersect(u, v).in(w);
}

void build(vector <Plane> planes) {
    int head = 0;
    int tail = 0;
    for (int i = 0; i < (int)planes.size(); ++ i) {
        while (tail - head > 1 && !check(queue[tail - 2], queue[tail - 1], planes[i])) {
            tail --;
        }
        while (tail - head > 1 && !check(queue[head + 1], queue[head], planes[i])) {
            head ++;
        }
        queue[tail ++] = planes[i];
    }
    while (tail - head > 2 && !check(queue[tail - 2], queue[tail - 1], queue[head])) {
        tail --;
    }
    while (tail - head > 2 && !check(queue[head + 1], queue[head], queue[tail - 1])) {
        head ++;
    }
}
```

# 2 数据结构

## 2.1 kd 树

```
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 Point::to(const Rectangle &r) const {
    const Point &p = *this;
    long long ret = 0;
    for (int i = 0; i < 2; ++ i) {
        ret += sqr(std::min(std::max(p[i], r.min[i]), r.max[i]) - p[i]);
    }
    return ret;
}

std::vector<int> order;
int separator[N << 1];
Rectangle rectangles[N << 1];

int get_id(int l, int r) {
    return l + r | l != r;
}

int pivot;

bool compare(int i, int j) {
    if (points[i][pivot] != points[j][pivot]) {
        return points[i][pivot] < points[j][pivot];
    }
    return i < j;
}

void build(int l, int r, int type) {
    int id = get_id(l, r);
    rectangles[id] = Rectangle();
    for (int i = l; i <= r; ++ i) {
        rectangles[id].add(points[order[i]]);
    }
    if (l < r) {
        int m = l + r >> 1;
        pivot = type;
        std::nth_element(order.begin() + l, order.begin() + m, order.begin() + r + 1, compare);
        separator[id] = order[m];
        build(l, m, type ^ 1);
        build(m + 1, r, type ^ 1);
    }
}

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

void query(int l, int r, int type) {
    const Point &p = points[n];
    int id = get_id(l, r);
    if (SIZE(answer) == 2 && p.to(rectangles[id]) > answer.top().first) {

```

```

        return;
    }
    if (l == r) {
        answer.push(std::make_pair((p - points[order[l]]).norm2(), order[l]));
        if (SIZE(answer) > 2) {
            answer.pop();
        }
    } else {
        int m = l + r >> 1;
        pivot = type;
        int dir = compare(seperator[id], n);
        if (dir) {
            query(l, m, type ^ 1);
        }
        query(m + 1, r, type ^ 1);
        if (!dir) {
            query(l, m, type ^ 1);
        }
    }
}
}

```

## 2.2 坚固的数据结构

(Joke from crazyb0y)

### 2.2.1 坚固的线段树

```

struct Node {
    int count;
    Node *left, *right;

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

    Node* insert(int l, int r, int k);
};

Node* null;

Node* Node::insert(int l, int r, int k) {
    if (k < l || r <= k) {
        return this;
    }
    if (l + 1 == r) {
        return new Node(this->count + 1, null, null);
    }
    int m = (l + r) >> 1;
    return new Node(this->count + 1,
        this->left->insert(l, m, k),
        this->right->insert(m, r, k));
}

int main() {
    // initialize
    null = new Node(0, NULL, NULL);
    null->left = null->right = null;
}

```

### 2.2.2 坚固的平衡树

```
struct Node;

typedef std::pair <Node*, Node*> Pair;

struct Node {
    int size;
    Node *left, *right;

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

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

    Pair split(int);
};

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

Node *null;

Node* merge(Node *p, Node *q) {
    if (p == null) {
        return q;
    }
    if (q == null) {
        return p;
    }
    if (random(p->size, q->size)) {
        p->right = merge(p->right, q);
        return p->update();
    }
    q->left = merge(p, q->left);
    return q->update();
}

Pair Node::split(int n) {
    if (this == null) {
        return std::make_pair(null, null);
    }
    if (n <= left->total) {
        Pair ret = left->split(n);
        left = null;
        return std::make_pair(ret.first, merge(ret.second, this->update()));
    }
    Pair ret = right->split(n - left->total);
    right = null;
    return std::make_pair(merge(this->update(), ret.first), ret.second);
}

int main() {
    // initialize
    null = new Node(0, 0);
    null->left = null->right = null;
}
```

```
}
```

## 2.3 后缀三姐妹

### 2.3.1 后缀数组

```
int n, m, count[N], rank[N], array[N], new_rank[N][2], new_array[N], height[N];
```

```
void construct(char* string, int n) {
    memset(count, 0, sizeof(count));
    for (int i = 0; i < n; ++ i) {
        count[(int)string[i]] ++;
    }
    for (int i = 0; i < 256; ++ i) {
        count[i + 1] += count[i];
    }
    for (int i = 0; i < n; ++ i) {
        rank[i] = count[(int)string[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(count, 0, sizeof(count));
        for (int i = 0; i < n; ++ i) {
            count[new_rank[i][1]] ++;
        }
        for (int i = 0; i < n; ++ i) {
            count[i + 1] += count[i];
        }
        for (int i = n - 1; i >= 0; -- i) {
            new_array[-- count[new_rank[i][1]]] = i;
        }
        memset(count, 0, sizeof(count));
        for (int i = 0; i < n; ++ i) {
            count[new_rank[i][0]] ++;
        }
        for (int i = 0; i < n; ++ i) {
            count[i + 1] += count[i];
        }
        for (int i = n - 1; i >= 0; -- i) {
            array[-- count[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
                    && string[i + length] == string[j + length]) {
                length ++;
            }
            height[rank[i]] = length;
        }
    }
}
```

```

        if (length) {
            length --;
        }
    }
}

```

### 2.3.2 后缀自动机

```

struct State {
    int length;
    State *parent;
    State* go[C];

    State(int length) : length(length), parent(NULL) {
        memset(go, NULL, sizeof(go));
        states.push_back(this);
    }

    State* extend(State* start, int token) {
        State *p = this;
        State *np = new State(length + 1);
        while (p && !p->go[token]) {
            p->go[token] = np;
            p = p->parent;
        }
        if (!p) {
            np->parent = start;
        } else {
            State *q = p->go[token];
            if (p->length + 1 == q->length) {
                np->parent = q;
            } else {
                State *nq = new State(p->length + 1);
                memcpy(nq->go, q->go, sizeof(q->go));
                nq->parent = q->parent;
                np->parent = q->parent = nq;
                while (p && p->go[token] == q) {
                    p->go[token] = nq;
                    p = p->parent;
                }
            }
        }
        return np;
    }
};

```

## 2.4 最长回文串 Manacher 算法

palindrome[i]是以  $\frac{i}{2}$  为对称中心的最长回文串长度

```

void manacher(char *text, int n) {
    palindrome[0] = 1;
    for (int i = 1, j = 0; i < n; ++i) {
        if (j + palindrome[j] <= i) {
            palindrome[i] = 0;
        } else {
            palindrome[i] = min(palindrome[(j << 1) - i], j + palindrome[j] - i);
        }
    }
}

```



```

        while (i - palindrome[i] >= 0 && i + palindrome[i] < n
               && text[i - palindrome[i]] == text[i + palindrome[i]]) {
            palindrome[i] ++;
        }
        if (i + palindrome[i] > j + palindrome[j]) {
            j = i;
        }
    }
}

```

### 3 图论

### 4 数论

#### 4.1 Millar-rabin

```

typedef long long LL;

bool test(LL n, LL b) {
    LL m = n - 1;
    LL counter = 0;
    while (~m & 1) {
        m >>= 1;
        counter ++;
    }
    LL ret = pow_mod(b, m, n);
    if (ret == 1 || ret == n - 1) {
        return true;
    }
    counter --;
    while (counter >= 0) {
        ret = multiply_mod(ret, ret, n);
        if (ret == n - 1) {
            return true;
        }
        counter --;
    }
    return false;
}

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

bool is_prime(LL n) {
    if (n < 2) {
        return false;
    }
    if (n < 4) {
        return true;
    }
    if (n == 3215031751LL) {
        return false;
    }
    for (int i = 0; i < 12 && BASE[i] < n; ++ i) {
        if (!test(n, BASE[i])) {
            return false;
        }
    }
}

```

```

    return true;
}

```

## 4.2 Polar Rho

```

typedef long long LL;

LL pollard_rho(LL n, LL seed) {
    LL x, y, head = 1, tail = 2;
    x = y = rand() % (n - 1) + 1;
    while (true) {
        x = multiply_mod(x, x, n);
        x = add_mod(x, seed, n);
        if (x == y) {
            return n;
        }
        LL d = gcd(abs(x - y), n);
        if (1 < d && d < n) {
            return d;
        }
        head ++;
        if (head == tail) {
            y = x;
            tail <= 1;
        }
    }
}

vector <LL> divisors;

void factorize(LL n) {
    if (n > 1) {
        if (is_prime(n)) {
            divisors.push_back(n);
        } else {
            LL d = n;
            while (d >= n) {
                d = pollard_rho(n, rand() % (n - 1) + 1);
            }
            factorize(n / d);
            factorize(d);
        }
    }
}

```

## 4.3 快速傅里叶变换

```

void FFT(Complex P[], int n, int oper)
{
    for (int i = 1, j = 0; i < n - 1; i++) {
        for (int s = n; j ^= s >= 1, ~j & s;);
        if (i < j) {
            swap(P[i], P[j]);
        }
    }
    Complex unit_p0;
    for (int d = 0; (1 <= d) < n; d++) {
        int m = 1 << d, m2 = m * 2;

```

```

double p0 = pi / m * oper;
sincos(p0, &unit_p0.y, &unit_p0.x);
for (int i = 0; i < n; i += m2) {
    Complex unit = 1;
    for (int j = 0; j < m; j++) {
        Complex &P1 = P[i + j + m], &P2 = P[i + j];
        Complex t = unit * P1;
        P1 = P2 - t;
        P2 = P2 + t;
        unit = unit * unit_p0;
    }
}
}
}

```

## 4.4 直线下格点统计

计算

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

$(n, m > 0, a, b \geq 0)$

```

typedef long long LL;
LL count(LL n, LL a, LL b, LL m) {
    if (b == 0) {
        return n * (a / m);
    }
    if (a >= m) {
        return n * (a / m) + count(n, a % m, b, m);
    }
    if (b >= m) {
        return (n - 1) * n / 2 * (b / m) + count(n, a, b % m, m);
    }
    return count((a + b * n) / m, (a + b * n) % m, m, b);
}

```

## 5 Miscellaneous

### 5.1 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;
}

```

## 5.2 二次剩余

解方程  $x^2 \equiv n \pmod{p}$  ( $p > 2$ ), 找  $a$  使得  $\omega = a^2 - n$  不是二次剩余, 则

$$x \equiv (a + \sqrt{\omega})^{\frac{p+1}{2}} \left( \bmod \frac{\mathbb{F}_p[x]}{(x^2 - \omega)} \right)$$

### 5.3 五边形数定理

设  $p(n)$  是  $n$  的拆分数, 有

$$p(n) = \sum_k (-1)^{k-1} p\left(n - \frac{k(3k-1)}{2}\right)$$

### 5.4 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

$$\cos a = \cos b \cos c + \sin b \sin c \cos A$$

正弦定理

$$\frac{\sin A}{\sin a} = \frac{\sin B}{\sin b} = \frac{\sin C}{\sin c}$$

三角形面积是  $A + B + C - \pi$

### 5.5 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

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

### 5.6 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

$$a_0 p_k + a_1 p_{k-1} + \cdots + a_{k-1} p_1 + k a_k = 0$$

特别地, 对于

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

有

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