代码库

上海交通大学

October 23, 2013

Contents

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 坚固的数据结构

(Joke from **crazyb0y**)

2.1.1 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;
}
```

2.1.2 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;
}
```

2.2 后缀三姐妹

2.2.1 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 --;
            }
        }
    }
}
```

2.2.2 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;
    }
};
```

2.3 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

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

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

```
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 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

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

5.2 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

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

5.3 牛顿恒等式

设

$$\prod_{i=1}^{n} (x - x_i) = a_n + a_{n-1}x + \cdots + a_1 x^{n-1} + a_0 x^n$$

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

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

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