

Standard Code Library

Shanghai Jiao Tong University

October, 2015

Contents

1	数论算法	5
1.1	快速数论变换	5
1.2	多项式求逆	6
1.3	中国剩余定理	6
1.4	Miller Rabin	7
1.5	Pollard Rho	7
1.6	坚固的逆元	8
1.7	直线下整点个数	8
2	数值算法	10
2.1	快速傅立叶变换	10
2.2	单纯形法求解线性规划	11
2.3	自适应辛普森	12
3	数据结构	14
3.1	Splay 普通操作版	14
3.2	Splay 区间操作版	14
3.3	坚固的 Treap	14
3.4	k-d 树	14
3.5	树链剖分	14
3.6	Link-Cut-Tree	14
4	图论	15
4.1	强连通分量	15
4.2	2-SAT 问题	16
4.3	二分图最大匹配	17
4.3.1	Hungary 算法	17
4.3.2	Hopcroft Karp 算法	17
4.4	二分图最大权匹配	19

4.5	最大流	20
4.6	上下界网络流	22
4.6.1	无源汇的上下界可行流	22
4.6.2	有源汇的上下界可行流	22
4.6.3	有源汇的上下界最大流	22
4.6.4	有源汇的上下界最小流	22
4.7	最小费用最大流	22
4.7.1	稀疏图	22
4.7.2	稠密图	24
4.8	一般图最大匹配	26
4.9	无向图全局最小割	28
4.10	有根树的同构	29
4.11	哈密尔顿回路 (ORE 性质的图)	30
5	字符串	33
5.1	模式串匹配	33
5.2	AC 自动机	33
5.3	后缀数组	34
5.4	广义后缀自动机	35
5.5	Manacher 算法	36
5.6	回文树	36
5.7	循环串最小表示	36
6	计算几何	37
6.1	二维基础	37
6.1.1	点类	37
6.1.2	凸包	38
6.1.3	半平面交	38
6.1.4	最近点对	39
6.2	三维基础	39
6.2.1	点类	39
6.2.2	凸包	40
6.3	多边形	43
6.3.1	判断点在多边形内部	43
6.3.2	多边形内整点计数	44
6.4	圆	44
6.4.1	最小覆盖圆	44
6.4.2	多边形与圆的交面积	45

7 其它	47
7.1 STL 使用方法	47
7.1.1 nth_element	47
7.1.2 next_permutation	47
8 数学公式	48
8.1 常用数学公式	48
8.1.1 求和公式	48
8.1.2 斐波那契数列	48
8.1.3 错排公式	49
8.1.4 莫比乌斯函数	49
8.1.5 Burnside 引理	49
8.1.6 五边形数定理	49
8.1.7 树的计数	49
8.1.8 欧拉公式	50
8.1.9 皮克定理	50
8.1.10 牛顿恒等式	50
8.2 平面几何公式	51
8.2.1 三角形	51
8.2.2 四边形	51
8.2.3 正 n 边形	52
8.2.4 圆	52
8.2.5 棱柱	52
8.2.6 棱锥	53
8.2.7 棱台	53
8.2.8 圆柱	53
8.2.9 圆锥	53
8.2.10 圆台	54
8.2.11 球	54
8.2.12 球台	54
8.2.13 球扇形	54
8.3 立体几何公式	55
8.3.1 球面三角公式	55
8.3.2 四面体体积公式	55

Chapter 1

数论算法

1.1 快速数论变换

使用条件及注意事项: mod 必须要是一个形如 $a2^b + 1$ 的数, pri 表示 mod 的原根。

```
const int mod = 998244353;
const int pri = 3;
int prepare(int n) {
    int len = 1;
    for (; len <= 2 * n; len <<= 1);
    for (int i = 0; i <= len; i++) {
        e[0][i] = fpm(pri, (mod - 1) / len * i, mod);
        e[1][i] = fpm(pri, (mod - 1) / len * (len - i), mod);
    }
    return len;
}

void DFT(int *a, int n, int f) {
    for (int i = 0, j = 0; i < n; i++) {
        if (i > j) std::swap(a[i], a[j]);
        for (int t = n >> 1; (j ^= t) < t; t >>= 1);
    }
    for (int i = 2; i <= n; i <<= 1)
        for (int j = 0; j < n; j += i)
            for (int k = 0; k < (i >> 1); k++) {
                int A = a[j + k];
                int B = (long long)a[j + k + (i >> 1)] * e[f][n / i * k] % mod;
                a[j + k] = (A + B) % mod;
                a[j + k + (i >> 1)] = (A - B + mod) % mod;
            }
    if (f == 1) {
        long long rev = fpm(n, mod - 2, mod);
        for (int i = 0; i < n; i++) {
            a[i] = (long long)a[i] * rev % mod;
        }
    }
}
```

```

    }
}

```

1.2 多项式求逆

使用条件及注意事项：求一个多项式在模意义下的逆元。

```

void getInv(int *a, int *b, int n) {
    static int tmp[MAXN];
    std::fill(b, b + n, 0);
    b[0] = fpm(a[0], mod - 2, mod);
    for (int c = 1; c <= n; c <<= 1) {
        for (int i = 0; i < c; i++) tmp[i] = a[i];
        std::fill(b + c, b + (c << 1), 0);
        std::fill(tmp + c, tmp + (c << 1), 0);
        DFT(tmp, c << 1, 0);
        DFT(b, c << 1, 0);
        for (int i = 0; i < (c << 1); i++) {
            b[i] = (long long)(2 - (long long)tmp[i] * b[i] % mod + mod) * b[i] % mod;
        }
        DFT(b, c << 1, 1);
        std::fill(b + c, b + (c << 1), 0);
    }
}

```

1.3 中国剩余定理

使用条件及注意事项：模数可以不互质。

```

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

```

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

```

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 坚固的逆元

```

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;
    }
}

```

1.7 直线下整点个数

```

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);
}

```


}

Chapter 2

数值算法

2.1 快速傅立叶变换

```
int prepare(int n) {
    int len = 1;
    for (; len <= 2 * n; len <= 1);
    for (int i = 0; i < len; i++) {
        e[0][i] = Complex(cos(2 * pi * i / len), sin(2 * pi * i / len));
        e[1][i] = Complex(cos(2 * pi * i / len), -sin(2 * pi * i / len));
    }
    return len;
}

void DFT(Complex *a, int n, int f) {
    for (int i = 0, j = 0; i < n; i++) {
        if (i > j) std::swap(a[i], a[j]);
        for (int t = n >> 1; (j ^= t) < t; t >>= 1);
    }
    for (int i = 2; i <= n; i <= 1)
        for (int j = 0; j < n; j += i)
            for (int k = 0; k < (i >> 1); k++) {
                Complex A = a[j + k];
                Complex B = e[f][n / i * k] * a[j + k + (i >> 1)];
                a[j + k] = A + B;
                a[j + k + (i >> 1)] = A - B;
            }
    if (f == 1) {
        for (int i = 0; i < n; i++)
            a[i].a /= n;
    }
}
```

2.2 单纯形法求解线性规划

使用条件及注意事项：返回结果为 $\max\{c_{1 \times m} \cdot x_{m \times 1} \mid x_{m \times 1} \geq 0_{m \times 1}, a_{n \times m} \cdot x_{m \times 1} \leq b_{n \times 1}\}$

```
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;
}

```

2.3 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));
}
```

Chapter 3

数据结构

3.1 Splay 普通操作版

3.2 Splay 区间操作版

3.3 坚固的 Treap

3.4 k-d 树

3.5 树链剖分

3.6 Link-Cut-Tree

Chapter 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]);
        }
    }
    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 2-SAT 问题

```

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;
}

```

4.3 二分图最大匹配

4.3.1 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;
}

```

4.3.2 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;
        }
    }
}

```

```
}
```

4.4 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;
}

```

4.5 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;
}

```

4.6 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

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

4.6.1 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: $S^* \rightarrow v$, 容量为 $B(u, v)$; $u \rightarrow T^*$, 容量为 $B(u, v)$; $u \rightarrow v$, 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 $G(u, v) + B(u, v)$ 。

4.6.2 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 $T \rightarrow S$ 边上的流量。

4.6.3 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 ∞ , 下界为 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 $S^* \rightarrow T^*$ 的最大流, 再将从汇点 T 到源点 S 的这条边拆掉, 求一次 $S \rightarrow T$ 的最大流即可。

4.6.4 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 $S^* \rightarrow T^*$ 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 $S^* \rightarrow T^*$ 的最大流。若超级源点 S^* 出发的边全部满流, 则 $T \rightarrow S$ 边上的流量即为原图的最小流, 否则无解。

4.7 最小费用最大流

4.7.1 稀疏图

时间复杂度: $O(V \cdot E^2)$

```

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;
}

```

4.7.2 稠密图

使用条件：费用非负

时间复杂度： $O(V \cdot E^2)$

```

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;
}

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);
}

```

4.8 一般图最大匹配

时间复杂度: $O(V^3)$

```

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;
}

```

4.9 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;
}

```

4.10 有根树的同构

时间复杂度: $\mathcal{O}(V \log V)$

```

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++;
}
}

```

4.11 哈密尔顿回路（ORE 性质的图）

ORE 性质:

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

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;
}

```


Chapter 5

字符串

5.1 模式串匹配

```
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;
    }
}
```

5.2 AC 自动机

```
int size, c[MAXT][26], f[MAXT], fail[MAXT], d[MAXT];

int alloc() {
    size++;
    std::fill(c[size], c[size] + 26, 0);
    f[size] = fail[size] = d[size] = 0;
    return size;
}
```

```

void insert(char *s) {
    int len = strlen(s + 1), p = 1;
    for (int i = 1; i <= len; i++) {
        if (c[p][s[i] - 'a']) p = c[p][s[i] - 'a'];
        else{
            int newnode = alloc();
            c[p][s[i] - 'a'] = newnode;
            d[newnode] = s[i] - 'a';
            f[newnode] = p;
            p = newnode;
        }
    }
}

void buildfail() {
    static int q[MAXT];
    int left = 0, right = 0;
    fail[1] = 0;
    for (int i = 0; i < 26; i++) {
        c[0][i] = 1;
        if (c[1][i]) q[++right] = c[1][i];
    }
    while (left < right) {
        left++;
        int p = fail[f[q[left]]];
        while (!c[p][d[q[left]]]) p = fail[p];
        fail[q[left]] = c[p][d[q[left]]];
        for (int i = 0; i < 26; i++) {
            if (c[q[left]][i]) {
                q[++right] = c[q[left]][i];
            }
        }
    }
    for (int i = 1; i <= size; i++)
        for (int j = 0; j < 26; j++) {
            int p = i;
            while (!c[p][j]) p = fail[p];
            c[i][j] = c[p][j];
        }
}

```

5.3 后缀数组

```

namespace suffix_array{
    int wa[MAXN], wb[MAXN], ws[MAXN], wv[MAXN];
    bool cmp(int *r, int a, int b, int l) {
        return r[a] == r[b] && r[a + l] == r[b + l];
    }
}

```

```

}
void DA(int *r, int *sa, int n, int m) {
    int *x = wa, *y = wb, *t;
    for (int i = 0; i < m; i++) ws[i] = 0;
    for (int i = 0; i < n; i++) ws[x[i]] = r[i]++;
    for (int i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (int i = n - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
    for (int i, j = 1, p = 1; p < n; j <= 1, m = p) {
        for (p = 0, i = n - j; i < n; i++) y[p++] = i;
        for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
        for (i = 0; i < n; i++) wv[i] = x[y[i]];
        for (i = 0; i < m; i++) ws[i] = 0;
        for (i = 0; i < n; i++) ws[wv[i]]++;
        for (i = 1; i < m; i++) ws[i] += ws[i - 1];
        for (i = n - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
        for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
    }
}
void getheight(int *r, int *sa, int *rk, int *h, int n) {
    for (int i = 1; i <= n; i++) rk[sa[i]] = i;
    for (int i = 0, j, k = 0; i < n; h[rk[i++]] = k)
        for (k ? k-- : 0, j = sa[rk[i] - 1]; r[i + k] == r[j + k]; k++);
}
};

```

5.4 广义后缀自动机

```

void add(int x, int &last) {
    int lastnode = last;
    if (c[lastnode][x]) {
        int nownode = c[lastnode][x];
        if (l[nownode] == l[lastnode] + 1) last = nownode;
        else{
            int auxnode = ++size; l[auxnode] = l[lastnode] + 1;
            for (int i = 0; i < 26; i++) c[auxnode][i] = c[nownode][i];
            f[auxnode] = f[nownode]; f[nownode] = auxnode;
            for (; lastnode && c[lastnode][x] == nownode; lastnode = f[lastnode]) {
                c[lastnode][x] = auxnode;
            }
            last = auxnode;
        }
    }
    else{
        int newnode = ++size; l[newnode] = l[lastnode] + 1;
        for (; lastnode && !c[lastnode][x]; lastnode = f[lastnode]) c[lastnode][x] = newnode;
        if (!lastnode) f[newnode] = 1;
    }
}

```

```

    else{
        int nownode = c[lastnode][x];
        if (l[lastnode] + 1 == l[nownode]) f[newnode] = nownode;
        else{
            int auxnode = ++size; l[auxnode] = l[lastnode] + 1;
            for (int i = 0; i < 26; i++) c[auxnode][i] = c[nownode][i];
            f[auxnode] = f[nownode]; f[nownode] = f[newnode] = auxnode;
            for (; lastnode && c[lastnode][x] == nownode; lastnode = f[lastnode]) {
                c[lastnode][x] = auxnode;
            }
        }
    }
    last = newnode;
}
}

```

5.5 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;
        }
    }
}

```

5.6 回文树

```

struct Palindromic_Tree{
    int nTree, nStr, last, c[MAXT][26], fail[MAXT], r[MAXN], l[MAXN], s[MAXN];
    int allocate(int len) {
        l[nTree] = len;
        r[nTree] = 0;
        fail[nTree] = 0;
        memset(c[nTree], 0, sizeof(c[nTree]));
    }
}

```

```

        return nTree++;
    }
    void init() {
        nTree = nStr = 0;
        int newEven = allocate(0);
        int newOdd = allocate(-1);
        last = newEven;
        fail[newEven] = newOdd;
        fail[newOdd] = newEven;
        s[0] = -1;
    }
    void add(int x) {
        s[++nStr] = x;
        int nownode = last;
        while (s[nStr - 1[nownode] - 1] != s[nStr]) nownode = fail[nownode];
        if (!c[nownode][x]) {
            int newnode = allocate(l[nownode] + 2), &newfail = fail[newnode];
            newfail = fail[nownode];
            while (s[nStr - 1[newfail] - 1] != s[nStr]) newfail = fail[newfail];
            newfail = c[newfail][x];
            c[nownode][x] = newnode;
        }
        last = c[nownode][x];
        r[last]++;
    }
    void count() {
        for (int i = nTree - 1; i >= 0; i--) {
            r[fail[i]] += r[i];
        }
    }
}

```

5.7 循环串最小表示

Chapter 6

计算几何

6.1 二维基础

6.1.1 点类

```
struct Point{
    double x, y;
    Point() {}
    Point(double x, double y):x(x), y(y) {}
    Point operator +(const Point &p)const {
        return Point(x + p.x, y + p.y);
    }
    Point operator -(const Point &p)const {
        return Point(x - p.x, y - p.y);
    }
    Point operator *(const double &p)const {
        return Point(x * p, y * p);
    }
    Point operator /(const double &p)const {
        return Point(x / p, y / p);
    }
    int read() {
        return scanf("%lf%lf", &x, &y);
    }
};

struct Line{
    Point a, b;
    Line() {}
    Line(Point a, Point b):a(a), b(b) {}
};
```

6.1.2 凸包

```

bool Pair_Comp(const Point &a, const Point &b) {
    if (dcmp(a.x - b.x) < 0) return true;
    if (dcmp(a.x - b.x) > 0) return false;
    return dcmp(a.y - b.y) < 0;
}

int Convex_Hull(int n, Point *P, Point *C) {
    sort(P, P + n, Pair_Comp);
    int top = 0;
    for (int i = 0; i < n; i++) {
        while (top >= 2 && dcmp(det(C[top - 1] - C[top - 2], P[i] - C[top - 2])) <= 0) top--;
        C[top++] = P[i];
    }
    int lasttop = top;
    for (int i = n - 1; i >= 0; i--) {
        while (top > lasttop && dcmp(det(C[top - 1] - C[top - 2], P[i] - C[top - 2])) <= 0)
            top--;
        C[top++] = P[i];
    }
    return top;
}

```

6.1.3 半平面交

```

bool isOnLeft(const Point &x, const Line &l) {
    double d = det(x - l.a, l.b - l.a);
    return dcmp(d) <= 0;
}

// 传入一个线段的集合 L, 传出 A, 并且返回 A 的大小
int getIntersectionOfHalfPlane(int n, Line *L, Line *A) {
    Line *q = new Line[n + 1];
    Point *p = new Point[n + 1];
    sort(L, L + n, Polar_Angle_Comp_Line);
    int l = 1, r = 0;
    for (int i = 0; i < n; i++) {
        while (l < r && !isOnLeft(p[r - 1], L[i])) r--;
        while (l < r && !isOnLeft(p[l], L[i])) l++;
        q[++r] = L[i];
        if (l < r && is_Colinear(q[r], q[r - 1])) {
            r--;
            if (isOnLeft(L[i].a, q[r])) q[r] = L[i];
        }
        if (l < r) p[r - 1] = getIntersection(q[r - 1], q[r]);
    }
    while (l < r && !isOnLeft(p[r - 1], q[l])) r--;
}

```

```

    if (r - l + 1 <= 2) return 0;
    int tot = 0;
    for (int i = l; i <= r; i++) A[tot++] = q[i];
    return tot;
}

```

6.1.4 最近点对

6.2 三维基础

6.2.1 点类

```

int dcmp(const double &x) {
    return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1);
}

struct TPoint{
    double x, y, z;
    TPoint() {}
    TPoint(double x, double y, double z) : x(x), y(y), z(z) {}
    TPoint operator +(const TPoint &p) const {
        return TPoint(x + p.x, y + p.y, z + p.z);
    }
    TPoint operator -(const TPoint &p) const {
        return TPoint(x - p.x, y - p.y, z - p.z);
    }
    TPoint operator *(const double &p) const {
        return TPoint(x * p, y * p, z * p);
    }
    TPoint operator /(const double &p) const {
        return TPoint(x / p, y / p, z / p);
    }
    bool operator <(const TPoint &p) const {
        int dX = dcmp(x - p.x), dY = dcmp(y - p.y), dZ = dcmp(z - p.z);
        return dX < 0 || (dX == 0 && (dY < 0 || (dY == 0 && dZ < 0)));
    }
    bool read() {
        return scanf("%lf%lf%lf", &x, &y, &z) == 3;
    }
};

double sqrdist(const TPoint &a) {
    double ret = 0;
    ret += a.x * a.x;
    ret += a.y * a.y;
    ret += a.z * a.z;
    return ret;
}

```



```

}
double sqrdist(const TPoint &a, const TPoint &b) {
    double ret = 0;
    ret += (a.x - b.x) * (a.x - b.x);
    ret += (a.y - b.y) * (a.y - b.y);
    ret += (a.z - b.z) * (a.z - b.z);
    return ret;
}
double dist(const TPoint &a) {
    return sqrt(sqrdist(a));
}
double dist(const TPoint &a, const TPoint &b) {
    return sqrt(sqrdist(a, b));
}
TPoint det(const TPoint &a, const TPoint &b) {
    TPoint ret;
    ret.x = a.y * b.z - b.y * a.z;
    ret.y = a.z * b.x - b.z * a.x;
    ret.z = a.x * b.y - b.x * a.y;
    return ret;
}
double dot(const TPoint &a, const TPoint &b) {
    double ret = 0;
    ret += a.x * b.x;
    ret += a.y * b.y;
    ret += a.z * b.z;
    return ret;
}
double detdot(const TPoint &a, const TPoint &b, const TPoint &c, const TPoint &d) {
    return dot(det(b - a, c - a), d - a);
}

```

6.2.2 凸包

```

struct Triangle{
    TPoint a, b, c;
    Triangle() {}
    Triangle(TPoint a, TPoint b, TPoint c) : a(a), b(b), c(c) {}
    double getArea() {
        TPoint ret = det(b - a, c - a);
        return dist(ret) / 2.0;
    }
};

namespace Convex_Hull {
    struct Face{
        int a, b, c;
        bool isOnConvex;
        Face() {}
    }
}

```

```

    Face(int a, int b, int c) : a(a), b(b), c(c) {}
};

int nFace, left, right, whe[MAXN][MAXN];
Face queue[MAXF], tmp[MAXF];

bool isVisible(const std::vector<TPoint> &p, const Face &f, const TPoint &a) {
    return dcmp(detdot(p[f.a], p[f.b], p[f.c], a)) > 0;
}

bool init(std::vector<TPoint> &p) {
    bool check = false;
    for (int i = 1; i < (int)p.size(); i++) {
        if (dcmp(sqrdist(p[0], p[i]))) {
            std::swap(p[1], p[i]);
            check = true;
            break;
        }
    }
    if (!check) return false;
    check = false;
    for (int i = 2; i < (int)p.size(); i++) {
        if (dcmp(sqrdist(det(p[i] - p[0], p[1] - p[0])))) {
            std::swap(p[2], p[i]);
            check = true;
            break;
        }
    }
    if (!check) return false;
    check = false;
    for (int i = 3; i < (int)p.size(); i++) {
        if (dcmp(detdot(p[0], p[1], p[2], p[i]))) {
            std::swap(p[3], p[i]);
            check = true;
            break;
        }
    }
    if (!check) return false;
    for (int i = 0; i < (int)p.size(); i++)
        for (int j = 0; j < (int)p.size(); j++) {
            whe[i][j] = -1;
        }
    return true;
}

void pushface(const int &a, const int &b, const int &c) {
    nFace++;
    tmp[nFace] = Face(a, b, c);
    tmp[nFace].isOnConvex = true;
}

```

```

    whe[a][b] = nFace;
    whe[b][c] = nFace;
    whe[c][a] = nFace;
}

bool deal(const std::vector<TPoint> &p, const std::pair<int, int> &now, const TPoint &base
) {
    int id = whe[now.second][now.first];
    if (!tmp[id].isOnConvex) return true;
    if (isVisible(p, tmp[id], base)) {
        queue[++right] = tmp[id];
        tmp[id].isOnConvex = false;
        return true;
    }
    return false;
}

std::vector<Triangle> getConvex(std::vector<TPoint> &p) {
    static std::vector<Triangle> ret;
    ret.clear();
    if (!init(p)) return ret;
    if (!isVisible(p, Face(0, 1, 2), p[3])) pushface(0, 1, 2); else pushface(0, 2, 1);
    if (!isVisible(p, Face(0, 1, 3), p[2])) pushface(0, 1, 3); else pushface(0, 3, 1);
    if (!isVisible(p, Face(0, 2, 3), p[1])) pushface(0, 2, 3); else pushface(0, 3, 2);
    if (!isVisible(p, Face(1, 2, 3), p[0])) pushface(1, 2, 3); else pushface(1, 3, 2);
    for (int a = 4; a < (int)p.size(); a++) {
        TPoint base = p[a];
        for (int i = 1; i <= nFace; i++) {
            if (tmp[i].isOnConvex && isVisible(p, tmp[i], base)) {
                left = 0, right = 0;
                queue[++right] = tmp[i];
                tmp[i].isOnConvex = false;
                while (left < right) {
                    Face now = queue[++left];
                    if (!deal(p, std::make_pair(now.a, now.b), base)) pushface(now.a, now.
                        b, a);
                    if (!deal(p, std::make_pair(now.b, now.c), base)) pushface(now.b, now.
                        c, a);
                    if (!deal(p, std::make_pair(now.c, now.a), base)) pushface(now.c, now.
                        a, a);
                }
                break;
            }
        }
    }
    for (int i = 1; i <= nFace; i++) {
        Face now = tmp[i];
        if (now.isOnConvex) {
            ret.push_back(Triangle(p[now.a], p[now.b], p[now.c]));
        }
    }
}

```

```

        }
    }
    return ret;
}

};

int n;
std::vector<TPoint> p;
std::vector<Triangle> answer;

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        TPoint a;
        a.read();
        p.push_back(a);
    }
    answer = Convex_Hull::getConvex(p);
    double areaCounter = 0.0;
    for (int i = 0; i < (int)answer.size(); i++) {
        areaCounter += answer[i].getArea();
    }
    printf("%.3f\n", areaCounter);
    return 0;
}

```

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 多边形内整点计数

```

int getInside(int n, Point *P) { // 求多边形P内有多少个整数点
    int OnEdge = n;
    double area = getArea(n, P);
    for (int i = 0; i < n - 1; i++) {
        Point now = P[i + 1] - P[i];
        int y = (int)now.y, x = (int)now.x;
        OnEdge += abs(gcd(x, y)) - 1;
    }
    Point now = P[0] - P[n - 1];
    int y = (int)now.y, x = (int)now.x;
    OnEdge += abs(gcd(x, y)) - 1;
    double ret = area - (double)OnEdge / 2 + 1;
    return (int)ret;
}

```

6.4 圆

6.4.1 最小覆盖圆

```

Point getmid(Point a, Point b) {
    return Point((a.x + b.x) / 2, (a.y + b.y) / 2);
}
Point getcross(Point a, Point vA, Point b, Point vB) {
    Point u = a - b;
    double t = det(vB, u) / det(vA, vB);
    return a + vA * t;
}
Point getcir(Point a, Point b, Point c) {
    Point midA = getmid(a, b), vA = Point(-(b - a).y, (b - a).x);
    Point midB = getmid(b, c), vB = Point(-(c - b).y, (c - b).x);
    return getcross(midA, vA, midB, vB);
}
double mincir(Point *p, int n) {
    std::random_shuffle(p + 1, p + n + 1);
    Point O = p[1];
    double r = 0;
    for (int i = 2; i <= n; i++) {
        if (dist(O, p[i]) <= r) continue;
        O = p[i]; r = 0;
        for (int j = 1; j < i; j++) {

```

```

        if (dist(O, p[j]) <= r) continue;
        O = getmid(p[i], p[j]); r = dist(O, p[i]);
        for (int k = 1; k < j; k++) {
            if (dist(O, p[k]) <= r) continue;
            O = getcir(p[i], p[j], p[k]);
            r = dist(O, p[i]);
        }
    }
}
return r;
}

```

6.4.2 多边形与圆的交面积

```

// 求扇形面积
double getSectorArea(const Point &a, const Point &b, const double &r) {
    double c = (2.0 * r * r - sqrdist(a, b)) / (2.0 * r * r);
    double alpha = acos(c);
    return r * r * alpha / 2.0;
}

// 求二次方程  $ax^2 + bx + c = 0$  的解
std::pair<double, double> getSolution(const double &a, const double &b, const double &c) {
    double delta = b * b - 4.0 * a * c;
    if (dcmp(delta) < 0) return std::make_pair(0, 0);
    else return std::make_pair((-b - sqrt(delta)) / (2.0 * a), (-b + sqrt(delta)) / (2.0 * a));
}

// 直线与圆的交点
std::pair<Point, Point> getIntersection(const Point &a, const Point &b, const double &r) {
    Point d = b - a;
    double A = dot(d, d);
    double B = 2.0 * dot(d, a);
    double C = dot(a, a) - r * r;
    std::pair<double, double> s = getSolution(A, B, C);
    return std::make_pair(a + d * s.first, a + d * s.second);
}

// 原点到线段AB的距离
double getPointDist(const Point &a, const Point &b) {
    Point d = b - a;
    int sA = dcmp(dot(a, d)), sB = dcmp(dot(b, d));
    if (sA * sB <= 0) return det(a, b) / dist(a, b);
    else return std::min(dist(a), dist(b));
}

// a和b和原点组成的三角形与半径为r的圆的交的面积
double getArea(const Point &a, const Point &b, const double &r) {
    double dA = dot(a, a), dB = dot(b, b), dC = getPointDist(a, b), ans = 0.0;
    if (dcmp(dA - r * r) <= 0 && dcmp(dB - r * r) <= 0) return det(a, b) / 2.0;
    Point tA = a / dist(a) * r;

```

```

Point tB = b / dist(b) * r;
if (dcmp(dC - r) > 0) return getSectorArea(tA, tB, r);
std::pair<Point, Point> ret = getIntersection(a, b, r);
if (dcmp(dA - r * r) > 0 && dcmp(dB - r * r) > 0) {
    ans += getSectorArea(tA, ret.first, r);
    ans += det(ret.first, ret.second) / 2.0;
    ans += getSectorArea(ret.second, tB, r);
    return ans;
}
if (dcmp(dA - r * r) > 0) return det(ret.first, b) / 2.0 + getSectorArea(tA, ret.first, r)
;
else return det(a, ret.second) / 2.0 + getSectorArea(ret.second, tB, r);
}
// 求圆与多边形的交的主过程
double getArea(int n, Point *p, const Point &c, const double r) {
    double ret = 0.0;
    for (int i = 0; i < n; i++) {
        int sgn = dcmp(det(p[i] - c, p[(i + 1) % n] - c));
        if (sgn > 0) ret += getArea(p[i] - c, p[(i + 1) % n] - c, r);
        else ret -= getArea(p[(i + 1) % n] - c, p[i] - c, r);
    }
    return fabs(ret);
}

```

Chapter 7

其它

7.1 STL 使用方法

7.1.1 nth_element

用法: `nth_element(a + 1, a + id, a + n + 1);`

作用: 将排名为 *id* 的元素放在第 *id* 个位置。

7.1.2 next_permutation

用法: `next_permutation(a + 1, a + n + 1);`

作用: 以 *a* 中从小到大排序后为第一个排列, 求得当期数组 *a* 中的下一个排列, 返回值为当期排列是否为最后一个排列。

Chapter 8

数学公式

8.1 常用数学公式

8.1.1 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. $\sum_{k=1}^n k(k+1)(k+2)(k+3) = \frac{n(n+1)(n+2)(n+3)(n+4)}{5}$

8.1.2 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. $fib_m | fib_n^2 \Leftrightarrow n fib_n | m$

8.1.3 错排公式

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

8.1.4 莫比乌斯函数

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

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

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

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

8.1.5 Burnside 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:

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

8.1.6 五边形数定理

设 $p(n)$ 是 n 的拆分数, 有

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

8.1.7 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

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

当 n 为偶数时， n 个结点的无根树的个数为

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

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 $\mathbf{A}[G]$ 为图 G 的邻接矩阵、 $\mathbf{D}[G]$ 为图 G 的度数矩阵，则图 G 的不同生成树的个数为 $\mathbf{C}[G] = \mathbf{D}[G] - \mathbf{A}[G]$ 的任意一个 $n-1$ 阶主子式的行列式值。

8.1.8 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：

$$V - E + F = 2$$

8.1.9 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：

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

8.1.10 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

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

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

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

8.2 平面几何公式

8.2.1 三角形

1. 半周长

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

2. 面积

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

3. 中线

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

4. 角平分线

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

5. 高线

$$H_a = b \sin C = c \sin B = \sqrt{b^2 - \left(\frac{a^2 + b^2 - c^2}{2a}\right)^2}$$

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

$$R = \frac{abc}{4S} = \frac{a}{2\sin A} = \frac{b}{2\sin B} = \frac{c}{2\sin C}$$

8.2.2 四边形

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

$$1. a^2 + b^2 + c^2 + d^2 = D_1^2 + D_2^2 + 4M^2$$

$$2. S = \frac{1}{2} D_1 D_2 \sin A$$

3. 对于圆内接四边形

$$ac + bd = D_1 D_2$$

4. 对于圆内接四边形

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

8.2.3 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

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

8.2.4 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

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

8.2.5 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

$$T = S + 2A$$

8.2.6 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

$$T = S + 2A$$

8.2.7 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

$$T = S + A_1 + A_2$$

8.2.8 圆柱

1. 侧面积

$$S = 2\pi rh$$

2. 全面积

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

3. 体积

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

8.2.9 圆锥

1. 母线

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

2. 侧面积

$$S = \pi rl$$

3. 全面积

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

4. 体积

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

8.2.10 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

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

8.2.11 球

1. 全面积

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

2. 体积

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

8.2.12 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

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

8.2.13 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

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

8.3 立体几何公式

8.3.1 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

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

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 $A + B + C - \pi$

8.3.2 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

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