代码库

上海交通大学

September 1, 2015

Contents

1 数论 4
- 1.1 快速求逆元 4
- 1.2 扩展欧几里德算法 4
- 1.3 中国剩余定理 4
- 1.4 Miller Rabin 素数测试 4
- 1.5 Pollard Rho 大数分解 4
- 1.6 快速数论变换 5
- 1.7 原根 5
- 1.8 离散对数 5
- 1.9 离散平方根 5
- 1.10 佩尔方程求解 5
- 1.11 牛顿迭代法 5
- 1.12 直线下整点个数 5

2 数值 5
- 2.1 高斯消元 5
- 2.2 快速傅立叶变换 5
- 2.3 单纯形法求解线性规划 6
- 2.4 自适应辛普森 8
- 2.5 多项式方程求解 8
- 2.6 最小二乘法 8

3 数据结构 8
- 3.1 平衡的二叉查找树 8
  - 3.1.1 Treap 8
  - 3.1.2 Splay 8
- 3.2 坚固的数据结构 8
  - 3.2.1 坚固的线段树 8
  - 3.2.2 坚固的平衡树 9
  - 3.2.3 坚固的字符串 9
  - 3.2.4 坚固的左偏树 9
- 3.3 树上的魔术师 9
  - 3.3.1 轻重树链剖分 9
  - 3.3.2 Link Cut Tree 9
  - 3.3.3 AAA Tree 9
- 3.4 k-d 树 9

4 图论 9
- 4.1 强连通分量 9
- 4.2 双连通分量 9
  - 4.2.1 点双连通分量 9
  - 4.2.2 边双连通分量 9
- 4.3 2-SAT 问题 9
- 4.4 二分图最大匹配 9
  - 4.4.1 Hungary 算法 9

4.4.2 Hopcroft Karp 算法 . . . 10
4.5 二分图最大权匹配 . . . 11
4.5.1 KM 算法 . . . 11
4.5.2 扩展 KM 算法 . . . 12
4.6 最大流 . . . 12
4.7 最小费用最大流 . . . 13
4.7.1 稀疏图 . . . 13
4.7.2 稠密图 . . . 15
4.8 一般图最大匹配 . . . 16
4.9 无向图全局最小割 . . . 18
4.10 最小树形图 . . . 19
4.11 有根树的同构 . . . 19
4.12 度限制生成树 . . . 20
4.13 弦图相关 . . . 20
4.13.1 弦图的判定 . . . 20
4.13.2 弦图的团数 . . . 20
4.14 哈密尔顿回路（ORE 性质的图） . . . 20

5 字符串 22
5.1 模式匹配 . . . 22
5.1.1 KMP 算法 . . . 22
5.1.2 扩展 KMP 算法 . . . 22
5.1.3 AC 自动机 . . . 22
5.2 后缀三姐妹 . . . 22
5.2.1 后缀数组 . . . 22
5.2.2 后缀自动机 . . . 22
5.3 回文三兄弟 . . . 23
5.3.1 Manacher 算法 . . . 23
5.3.2 回文树 . . . 23
5.4 循环串最小表示 . . . 24

6 计算几何 26
6.1 二维基础 . . . 26
6.1.1 点类 . . . 26
6.1.2 凸包 . . . 26
6.1.3 半平面交 . . . 26
6.2 三维基础 . . . 26
6.2.1 点类 . . . 26
6.2.2 凸包 . . . 26
6.2.3 绕轴旋转 . . . 26
6.3 多边形 . . . 26
6.3.1 判断点在多边形内部 . . . 26
6.3.2 旋转卡壳 . . . 26
6.3.3 动态凸包 . . . 26
6.3.4 点到凸包的切线 . . . 26
6.3.5 直线与凸包的交点 . . . 26
6.3.6 凸多边形的交集 . . . 26
6.3.7 凸多边形内的最大圆 . . . 26
6.4 圆 . . . 26
6.4.1 圆类 . . . 26
6.4.2 圆的交集 . . . 26
6.4.3 最小覆盖圆 . . . 26
6.4.4 最小覆盖球 . . . 26
6.4.5 判断圆存在交集 . . . 26
6.4.6 圆与多边形的交集 . . . 26
6.5 三角形 . . . 26
6.5.1 三角形的内心 . . . 26
6.5.2 三角形的外心 . . . 26

6.5.3 三角形的垂心 26
6.6 黑暗科技 26
6.6.1 平面图形的转动惯量 26
6.6.2 平面区域处理 26
6.6.3 Vonoroi 图 26

7 其他 26
7.1 某年某月某日是星期几 26

8 数学 27
8.1 常用积分表 27
8.2 常用数学公式 27
8.3 平面几何公式 27
8.3.1 三角形 27
8.3.2 四边形 27
8.3.3 正 n 边形 27
8.3.4 圆 27
8.3.5 棱柱 27
8.3.6 棱锥 27
8.3.7 棱台 27
8.3.8 圆柱 27
8.3.9 圆锥 27
8.3.10 圆台 27
8.4 常用数表 27
8.4.1 梅森数 27

1 数论

1.1 快速求逆元

1.2 扩展欧几里德算法

1.3 中国剩余定理

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

```
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

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 牛顿迭代法

1.12 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);
}
```

2 数值

2.1 高斯消元

2.2 快速傅立叶变换

```
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

```
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 多项式方程求解

2.6 最小二乘法

3 数据结构

3.1 平衡的二叉查找树

3.1.1 Treap

3.1.2 Splay

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 value);
    int query(int l, int r, int qx);
};

Node* null;

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

```cpp
    return new Node(this->left->modify(l, mid, ql, qr, value),
                    this->right->modify(mid + 1, r, ql, qr, value),
                    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

3.2.3 坚固的字符串

3.2.4 坚固的左偏树

3.3 树上的魔术师

3.3.1 轻重树链剖分

3.3.2 Link Cut Tree

3.3.3 AAA Tree

3.4 k-d 树

4 图论

4.1 强连通分量

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

4.4 二分图最大匹配

4.4.1 Hungary 算法

```cpp
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

```
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

4.5.1 KM 算法

```
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;
}
```

4.5.2 扩展 KM 算法

4.6 最大流

```
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 最小费用最大流

4.7.1 稀疏图

```
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

```
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

```
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

```
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[
                }
                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 最小树形图

4.11 有根树的同构

```
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++;
    }
}
```

4.12 度限制生成树

4.13 弦图相关

4.13.1 弦图的判定

4.13.2 弦图的团数

4.14 哈密尔顿回路（ORE 性质的图）

```
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

5.1.3 AC 自动机

5.2 后缀三姐妹

5.2.1 后缀数组

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

6.1.3 半平面交

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

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 凸多边形的交集

6.3.7 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

6.6 黑暗科技

6.6.1 平面图形的转动惯量

6.6.2 平面区域处理

6.6.3 Vonoroi 图

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)
    }
    return answer;
}
```

8 数学

8.1 常用积分表

8.2 常用数学公式

8.3 平面几何公式

8.3.1 三角形

8.3.2 四边形

8.3.3 正 n 边形

8.3.4 圆

8.3.5 棱柱

8.3.6 棱锥

8.3.7 棱台

8.3.8 圆柱

8.3.9 圆锥

8.3.10 圆台

8.4 常用数表

8.4.1 梅森数