Fantasy Legend

For Manual/Intelligence

August 6, 2018

Standard Code Library

Shanghai Jiao Tong University

October, 2015

Contents

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，prt 表示 mod 的原根。

```
const int mod = 998244353;
const int prt = 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(prt, (mod - 1) / len * i, mod);
        e[1][i] = fpm(prt, (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

使用条件及注意事项:

1. 插入 x 数
2. 删除 x 数 (若有多个相同的数，因只删除一个)
3. 查询 x 数的排名 (若有多个相同的数，因输出最小的排名)
4. 查询排名为 x 的数
5. 求 x 的前驱 (前驱定义为小于 x，且最大的数)
6. 求 x 的后继 (后继定义为大于 x，且最小的数)

```
int pred(int x) {
    splay(x, -1);
    for (x = c[x][0]; c[x][1]; x = c[x][1]);
    return x;
}
int succ(int x) {
    splay(x, -1);
    for (x = c[x][1]; c[x][0]; x = c[x][0]);
    return x;
}
void remove(int x) {
    if (b[x] > 1) {b[x]--; splay(x, -1); return;}
    splay(x, -1);
    if (!c[x][0] && !c[x][1]) rt = 0;
    else if (c[x][0] && !c[x][1]) f[rt = c[x][0]] = -1;
    else if (!c[x][0] && c[x][1]) f[rt = c[x][1]] = -1;
    else{
        int t = pred(x); f[rt = c[x][0]] = -1;
        c[t][1] = c[x][1]; f[c[x][1]] = t;
        splay(c[x][1], -1);
    }
```

```
    c[x][0] = c[x][1] = f[x] = d[x] = s[x] = b[x] = 0;
}
int find(int z) {
    int x=rt;
    while (d[x]!=z)
        if (c[x][d[x]<z]) x=c[x][d[x]<z];
        else break;
    return x;
}
void insert(int z) {
    if (!rt) {
        f[rt = ++size] = -1;
        d[size] = z; b[size] = 1;
        splay(size, -1);
        return;
    }
    int x = find(z);
    if (d[x] == z) {
        b[x]++;
        splay(x, -1);
        return;
    }
    c[x][d[x]<z] = ++size; f[size] = x;
    d[size] = z; b[size] = s[size] = 1;
    splay(size, -1);
}
int select(int z) {
    int x = rt;
    while (z < s[c[x][0]] + 1 || z > s[c[x][0]] + b[x])
        if (z > s[c[x][0]] + b[x]) {
            z -= s[c[x][0]] + b[x];
            x = c[x][1];
        }
        else x = c[x][0];
    return x;
}
int main() {
    scanf("%d",&n);
    for (int i = 1; i <= n; i++) {
        int opt, x;
        scanf("%d%d", &opt, &x);
        if (opt == 1) insert(x);
        else if (opt == 2) remove(find(x)); //删除x数(若有多个相同的数，因只删除一个)
        else if (opt == 3) { // 查询x数的排名(若有多个相同的数，因输出最小的排名)
            insert(x);
            printf("%d\n", s[c[find(x)][0]] + 1);
            remove(find(x));
        }
        else if (opt == 4) printf("%d\n",d[select(x)]);
        else if (opt == 5) {
            insert(x);
            printf("%d\n", d[pred(find(x))]);
            remove(find(x));
```

```
        }
        else if (opt == 6) {
            insert(x);
            printf("%d\n", d[succ(find(x))]);
            remove(find(x));
        }
    }
    return 0;
}
```

3.2 Splay 区间操作版

使用条件及注意事项:

这是为 NOI2005 维修数列的代码，仅供区间操作用的 splay 参考。

```
const int INF = 100000000;
const int Maxspace = 500000;
struct SplayNode{
    int ls, rs, zs, ms;
    SplayNode() {
        ms = 0;
        ls = rs = zs = -INF;
    }
    SplayNode(int d) {
        ms = zs = ls = rs = d;
    }
    SplayNode operator +(const SplayNode &p)const {
        SplayNode ret;
        ret.ls = max(ls, ms + p.ls);
        ret.rs = max(rs + p.ms, p.rs);
        ret.zs = max(rs + p.ls, max(zs, p.zs));
        ret.ms = ms + p.ms;
        return ret;
    }
}t[MAXN], d[MAXN];
int n, m, rt, top, a[MAXN], f[MAXN], c[MAXN][2], g[MAXN], h[MAXN], z[MAXN];
bool r[MAXN], b[MAXN];
void makesame(int x, int s) {
    if (!x) return;
    b[x] = true;
    d[x] = SplayNode(g[x] = s);
    t[x].zs = t[x].ms = g[x] * h[x];
    t[x].ls = t[x].rs = max(g[x], g[x] * h[x]);
}
void makerev(int x) {
    if (!x) return;
    r[x] ^= 1;
    swap(c[x][0], c[x][1]);
    swap(t[x].ls, t[x].rs);
}
void pushdown(int x) {
    if (!x) return;
```

```
    if (r[x]) {
        makerev(c[x][0]);
        makerev(c[x][1]);
        r[x]=0;
    }
    if (b[x]) {
        makesame(c[x][0],g[x]);
        makesame(c[x][1],g[x]);
        b[x]=g[x]=0;
    }
}
void updata(int x) {
    if (!x) return;
    h[x]=h[c[x][0]]+h[c[x][1]]+1;
    t[x]=t[c[x][0]]+d[x]+t[c[x][1]];
}
void rotate(int x,int k) {
    pushdown(x);pushdown(c[x][k]);
    int y = c[x][k]; c[x][k] = c[y][k^1]; c[y][k^1] = x;
    if (f[x] != -1) c[f[x]][c[f[x]][1] == x] = y; else rt = y;
    f[y] = f[x]; f[x] = y; f[c[x][k]] = x;
    updata(x); updata(y);
}
void splay(int x, int s) {
    while (f[x] != s) {
        if (f[f[x]]!=s) {
            pushdown(f[f[x]]);
            rotate(f[f[x]], (c[f[f[x]]][1] == f[x]) ^ r[f[f[x]]]);
        }
        pushdown(f[x]);
        rotate(f[x], (c[f[x]][1]==x) ^ r[f[x]]);
    }
}
void build(int &x,int l,int r) {
    if (l > r) {x = 0; return;}
    x = z[top--];
    if (l < r) {
        build(c[x][0],l,(l+r>>1)-1);
        build(c[x][1],(l+r>>1)+1,r);
    }
    f[c[x][0]] = f[c[x][1]] = x;
    d[x] = SplayNode(a[l+r>>1]);
    updata(x);
}
void init() {
    d[0] = SplayNode();
    f[rt=2] = -1;
    f[1] = 2; c[2][0] = 1;
    int x;
    build(x,1,n);
    c[1][1] = x; f[x] = 1;
    splay(x, -1);
}
```

```
int find(int z) {
    int x = rt; pushdown(x);
    while (z != h[c[x][0]] + 1) {
        if (z > h[c[x][0]] + 1) {
            z -= h[c[x][0]] + 1;
            x = c[x][1];
        }
        else x = c[x][0];
        pushdown(x);
    }
    return x;
}
void getrange(int &x,int &y) {
    y = x + y - 1;
    x = find(x);
    y = find(y + 2);
    splay(y, -1);
    splay(x, y);
}
void recycle(int x) {
    if (!x) return;
    recycle(c[x][0]);
    recycle(c[x][1]);
    z[++top]=x;
    t[x] = d[x] = SplayNode();
    r[x] = b[x] = g[x] = f[x] = h[x] = 0;
    c[x][0] = c[x][1]=0;
}
int main() {
    scanf("%d%d",&n,&m);
    for (int i = 1; i <= n; i++) scanf("%d",a+i);
    for (int i = Maxspace; i>=3; i--) z[++top] = i;
    init();
    for (int i = 1; i <= m; i++) {
        char op[10];
        int x, y, tmp;
        scanf("%s", op);
        if (!strcmp(op, "INSERT")) {
            scanf("%d%d", &x, &y);
            n += y;
            if (!y) continue;
            for (int i = 1; i <= y; i++) scanf("%d",a+i);
            build(tmp, 1, y);
            x = find(x + 1); pushdown(x);
            if (!c[x][1]) {c[x][1] = tmp; f[tmp] = x;}
            else{
                x = c[x][1]; pushdown(x);
                while (c[x][0]) {
                    x = c[x][0];
                    pushdown(x);
                }
                c[x][0] = tmp; f[tmp] = x;
            }
```

```
            splay(tmp, -1);
        }
        else if (!strcmp(op, "DELETE")) {
            scanf("%d%d", &x, &y); n -= y;
            if (!y) continue;
            getrange(x, y);
            int k = (c[y][0] == x);
            recycle(c[x][k]);
            f[c[x][k]] = 0;
            c[x][k] = 0;
            splay(x, -1);
        }
        else if (!strcmp(op, "REVERSE")) {
            scanf("%d%d", &x, &y);
            if (!y) continue;
            getrange(x, y);
            int k = (c[y][0]==x);
            makerev(c[x][k]);
            splay(c[x][k], -1);
        }
        else if (!strcmp(op, "GET-SUM")) {
            scanf("%d%d", &x, &y);
            if (!y) {
                printf("0\n");
                continue;
            }
            getrange(x,y);
            int k = (c[y][0] == x);
            printf("%d\n", t[c[x][k]].ms);
            splay(c[x][k], -1);
        }
        else if (!strcmp(op, "MAX-SUM")) {
            x = 1; y = n;
            getrange(x, y);
            int k = (c[y][0] == x);
            printf("%d\n", t[c[x][k]].zs);
            splay(c[x][k], -1);
        }
        else if (!strcmp(op, "MAKE-SAME")) {
            scanf("%d%d%d", &x, &y, &tmp);
            if (!y) continue;
            getrange(x, y);
            int k = (c[y][0] == x);
            makesame(c[x][k], tmp);
            splay(c[x][k], -1);
        }
    }
    return 0;
}
```

3.3 坚固的 Treap

使用条件及注意事项：题目来源 UVA 12358

```
int ran() {
    static int ret = 182381727;
    return (ret += (ret << 1) + 717271723) & (~0u >> 1);
}

int alloc(int node = 0) {
    size++;
    if (node) {
        c[size][0] = c[node][0];
        c[size][1] = c[node][1];
        s[size] = s[node];
        d[size] = d[node];
    }
    else{
        c[size][0] = 0;
        c[size][1] = 0;
        s[size] = 1;
        d[size] = '␣';
    }
    return size;
}

void update(int x) {
    s[x] = 1;
    if (c[x][0]) s[x] += s[c[x][0]];
    if (c[x][1]) s[x] += s[c[x][1]];
}

int merge(const std::pair<int, int> &a) {
    if (!a.first) return a.second;
    if (!a.second) return a.first;
    if (ran() % (s[a.first] + s[a.second]) < s[a.first]) {
        int newnode = alloc(a.first);
        c[newnode][1] = merge(std::make_pair(c[newnode][1], a.second));
        update(newnode);
        return newnode;
    }
    else{
        int newnode = alloc(a.second);
        c[newnode][0] = merge(std::make_pair(a.first, c[newnode][0]));
        update(newnode);
        return newnode;
    }
}

std::pair<int, int> split(int x, int k) {
    if (!x || !k) return std::make_pair(0, x);
    int newnode = alloc(x);
    if (k <= s[c[x][0]]) {
```

```
        std::pair<int, int> ret = split(c[newnode][0], k);
        c[newnode][0] = ret.second;
        update(newnode);
        return std::make_pair(ret.first, newnode);
    }
    else{
        std::pair<int, int> ret = split(c[newnode][1], k - s[c[x][0]] - 1);
        c[newnode][1] = ret.first;
        update(newnode);
        return std::make_pair(newnode, ret.second);
    }
}

void travel(int x) {
    if (c[x][0]) travel(c[x][0]);
    putchar(d[x]);
    if (d[x] == 'c') cnt++;
    if (c[x][1]) travel(c[x][1]);
}

int build(int l, int r) {
    int newnode = alloc();
    d[newnode] = tmp[l + r >> 1];
    if (l <= (l + r >> 1) - 1) c[newnode][0] = build(l, (l + r >> 1) - 1);
    if ((l + r >> 1) + 1 <= r) c[newnode][1] = build((l + r >> 1) + 1, r);
    update(newnode);
    return newnode;
}

int main() {
    scanf("%d", &n);
    for (int i = 1, last = 0; i <= n; i++) {
        int op, v, p, l;
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d%s", &p, tmp + 1);
            p -= cnt;
            std::pair<int, int> ret = split(rt[last], p);
            rt[last + 1] = merge(std::make_pair(ret.first, build(1, strlen(tmp + 1))));
            rt[last + 1] = merge(std::make_pair(rt[last + 1], ret.second));
            last++;
        }
        else if (op == 2) {
            scanf("%d%d", &p, &l);
            p -= cnt; l -= cnt;
            std::pair<int, int> A = split(rt[last], p - 1);
            std::pair<int, int> B = split(A.second, l);
            rt[last + 1] = merge(std::make_pair(A.first, B.second));
            last++;
        }
        else if (op == 3) {
            scanf("%d%d%d", &v, &p, &l);
            v -= cnt; p -= cnt; l -= cnt;
```

```
            std::pair<int, int> A = split(rt[v], p - 1);
            std::pair<int, int> B = split(A.second, l);
            travel(B.first);
            puts("");
        }
    }
    return 0;
}
```

3.4 k-d 树

使用条件及注意事项：这是求 k 远点的代码，要求 k 近点的话把堆的比较函数改一改，把朝左儿子或者是右儿子的方向改一改。

```
struct Heapnode{
    long long d;
    int pos;
    bool operator <(const Heapnode &p)const {
        return d > p.d || (d == p.d && pos < p.pos);
    }
};

struct MsgNode{
    int xmin, xmax, ymin, ymax;
    MsgNode() {}
    MsgNode(const Point &a) : xmin(a.x), xmax(a.x), ymin(a.y), ymax(a.y) {}
    long long dist(const Point &a) {
        int dx = std::max(std::abs(a.x - xmin), std::abs(a.x - xmax));
        int dy = std::max(std::abs(a.y - ymin), std::abs(a.y - ymax));
        return (long long)dx * dx + (long long)dy * dy;
    }
    MsgNode operator +(const MsgNode &rhs)const {
        MsgNode ret;
        ret.xmin = std::min(xmin, rhs.xmin);
        ret.xmax = std::max(xmax, rhs.xmax);
        ret.ymin = std::min(ymin, rhs.ymin);
        ret.ymax = std::max(ymax, rhs.ymax);
        return ret;
    }
};

struct TNode{
    int l, r;
    Point p;
    MsgNode d;
}tree[MAXN];

void buildtree(int &rt, int l, int r, int pivot) {
    if (l > r) return;
    rt = ++size;
    int mid = l + r >> 1;
    if (pivot == 1) std::nth_element(p + l, p + mid, p + r + 1, cmpx);
```

```
    if (pivot == 0) std::nth_element(p + l, p + mid, p + r + 1, cmpy);
    tree[rt].d = MsgNode(tree[rt].p = p[mid]);
    buildtree(tree[rt].l, l, mid - 1, pivot ^ 1);
    buildtree(tree[rt].r, mid + 1, r, pivot ^ 1);
    if (tree[rt].l) tree[rt].d = tree[rt].d + tree[tree[rt].l].d;
    if (tree[rt].r) tree[rt].d = tree[rt].d + tree[tree[rt].r].d;
}

void query(int rt, const Point &a, int k, int pivot) {
    Heapnode now = (Heapnode){dist(a, tree[rt].p), tree[rt].p.pos};
    if (heap.size() < k) heap.push(now);
    else if (now < heap.top()) {heap.pop(); heap.push(now);}
    int lson = tree[rt].l, rson = tree[rt].r;
    if (pivot == 1 && cmpx(a, tree[rt].p)) std::swap(lson, rson);
    if (pivot == 0 && cmpy(a, tree[rt].p)) std::swap(lson, rson);
    if (lson && (heap.size() < k || tree[lson].d.dist(a) >= heap.top().d)) query(lson, a, k,
        pivot ^ 1);
    if (rson && (heap.size() < k || tree[rson].d.dist(a) >= heap.top().d)) query(rson, a, k,
        pivot ^ 1);
}

int main() {
    for (int i = 1; i <= q; i++) {
        int k;
        Point now;
        now.read();
        scanf("%d", &k);
        while (!heap.empty()) heap.pop();
        query(rt, now, k, 1);
        printf("%d\n", heap.top().pos);
    }
    return 0;
}
```

3.5 树链剖分

3.5.1 点操作版本

使用条件及注意事项：树上最大（非空）子段和，注意一条路径询问的时候信息统计的顺序。

```
struct Node{
    int asum, lsum, rsum, zsum;
    Node() {
        asum = 0;
        lsum = -INF;
        rsum = -INF;
        zsum = -INF;
    }
    Node(int d) : asum(d), lsum(d), rsum(d), zsum(d) {}
    Node operator +(const Node &rhs)const {
        Node ret;
        ret.asum = asum + rhs.asum;
```

```
        ret.lsum = std::max(lsum, asum + rhs.lsum);
        ret.rsum = std::max(rsum + rhs.asum, rhs.rsum);
        ret.zsum = std::max(zsum, rhs.zsum);
        ret.zsum = std::max(ret.zsum, rsum + rhs.lsum);
        return ret;
    }
}tree[MAXN * 6];

int n, q, cnt, tot, h[MAXN], d[MAXN], t[MAXN], f[MAXN], s[MAXN], z[MAXN], w[MAXN], o[MAXN], a[MAXN];
std::pair<bool, int> flag[MAXN * 6];

void addedge(int x, int y) {
    cnt++; e[cnt] = (Edge){y, h[x]}; h[x] = cnt;
    cnt++; e[cnt] = (Edge){x, h[y]}; h[y] = cnt;
}

void makesame(int n, int l, int r, int d) {
    flag[n] = std::make_pair(true, d);
    tree[n].asum = d * (r - l + 1);
    if (d > 0) {
        tree[n].lsum = d * (r - l + 1);
        tree[n].rsum = d * (r - l + 1);
        tree[n].zsum = d * (r - l + 1);
    }
    else{
        tree[n].lsum = d;
        tree[n].rsum = d;
        tree[n].zsum = d;
    }
}

void pushdown(int n, int l, int r) {
    if (flag[n].first) {
        makesame(n << 1, l, l + r >> 1, flag[n].second);
        makesame(n << 1 ^ 1, (l + r >> 1) + 1, r, flag[n].second);
        flag[n] = std::make_pair(false, 0);
    }
}

void modify(int n, int l, int r, int x, int y, int d) {
    if (x <= l && r <= y) {
        makesame(n, l, r, d);
        return;
    }
    pushdown(n, l, r);
    if ((l + r >> 1) < x) modify(n << 1 ^ 1, (l + r >> 1) + 1, r, x, y, d);
    else if ((l + r >> 1) + 1 > y) modify(n << 1, l, l + r >> 1, x, y, d);
    else{
        modify(n << 1, l, l + r >> 1, x, y, d);
        modify(n << 1 ^ 1, (l + r >> 1) + 1, r, x, y, d);
    }
    tree[n] = tree[n << 1] + tree[n << 1 ^ 1];
```

```
}

Node query(int n, int l, int r, int x, int y) {
    if (x <= l && r <= y) return tree[n];
    pushdown(n, l, r);
    if ((l + r >> 1) < x) return query(n << 1 ^ 1, (l + r >> 1) + 1, r, x, y);
    else if ((l + r >> 1) + 1 > y) return query(n << 1, l, l + r >> 1, x, y);
    else{
        Node left = query(n << 1, l, l + r >> 1, x, y);
        Node right = query(n << 1 ^ 1, (l + r >> 1) + 1, r, x, y);
        return left + right;
    }
}

void modify(int x, int y, int val) {
    int fx = t[x], fy = t[y];
    while (fx != fy) {
        if (d[fx] > d[fy]) {
            modify(1, 1, n, w[fx], w[x], val);
            x = f[fx]; fx = t[x];
        }
        else{
            modify(1, 1, n, w[fy], w[y], val);
            y = f[fy]; fy = t[y];
        }
    }
    if (d[x] < d[y]) modify(1, 1, n, w[x], w[y], val);
    else modify(1, 1, n, w[y], w[x], val);
}

Node query(int x, int y) {
    int fx = t[x], fy = t[y];
    Node left = Node(), right = Node();
    while (fx != fy) {
        if (d[fx] > d[fy]) {
            left = query(1, 1, n, w[fx], w[x]) + left;
            x = f[fx]; fx = t[x];
        }
        else{
            right = query(1, 1, n, w[fy], w[y]) + right;
            y = f[fy]; fy = t[y];
        }
    }
    if (d[x] < d[y]) {
        right = query(1, 1, n, w[x], w[y]) + right;
    }
    else{
        left = query(1, 1, n, w[y], w[x]) + left;
    }
    std::swap(left.lsum, left.rsum);
    return left + right;
}

```

```
void predfs(int x) {
    s[x] = 1; z[x] = 0;
    for (int i = h[x]; i; i = e[i].next) {
        if (e[i].node == f[x]) continue;
        f[e[i].node] = x;
        d[e[i].node] = d[x] + 1;
        predfs(e[i].node);
        s[x] += s[e[i].node];
        if (s[z[x]] < s[e[i].node]) z[x] = e[i].node;
    }
}

void getanc(int x, int anc) {
    t[x] = anc; w[x] = ++tot; o[tot] = x;
    if (z[x]) getanc(z[x], anc);
    for (int i = h[x]; i; i = e[i].next) {
        if (e[i].node == f[x] || e[i].node == z[x]) continue;
        getanc(e[i].node, e[i].node);
    }
}

void buildtree(int n, int l, int r) {
    if (l == r) {
        tree[n] = Node(a[o[l]]);
        return;
    }
    buildtree(n << 1, l, l + r >> 1);
    buildtree(n << 1 ^ 1, (l + r >> 1) + 1, r);
    tree[n] = tree[n << 1] + tree[n << 1 ^ 1];
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", a + i);
    for (int i = 1; i < n; i++) {
        int x, y; scanf("%d%d", &x, &y);
        addedge(x, y);
    }
    predfs(1);
    getanc(1, 1);
    buildtree(1, 1, n);
    scanf("%d", &q);
    for (int i = 1; i <= q; i++) {
        int op, x, y, c;
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d%d", &x, &y);
            Node ret = query(x, y);
            printf("%d\n", std::max(0, ret.zsum));
        }
        else{
            scanf("%d%d%d", &x, &y, &c);
            modify(x, y, c);
```

```
        }
    }
    return 0;
}
```

3.5.2 链操作版本

```
void modify(int x, int y) {
    int fx = t[x], fy = t[y];
    while (fx != fy) {
        if (d[fx] > d[fy]) {
            modify(1, 1, n, w[fx], w[x]);
            x = f[fx]; fx = t[x];
        }
        else{
            modify(1, 1, n, w[fy], w[y]);
            y = f[fy]; fy = t[y];
        }
    }
    if (x != y) {
        if (d[x] < d[y]) modify(1, 1, n, w[z[x]], w[y]);
        else modify(1, 1, n, w[z[y]], w[x]);
    }
}
```

3.6 Link-Cut-Tree

```
struct MsgNode{
    int leftColor, rightColor, answer;
    MsgNode() {
        leftColor = -1;
        rightColor = -1;
        answer = 0;
    }
    MsgNode(int c) {
        leftColor = rightColor = c;
        answer = 1;
    }
    MsgNode operator +(const MsgNode &p)const {
        if (answer == 0) return p;
        if (p.answer == 0) return *this;
        MsgNode ret;
        ret.leftColor = leftColor;
        ret.rightColor = p.rightColor;
        ret.answer = answer + p.answer - (rightColor == p.leftColor);
        return ret;
    }
}d[MAXN], g[MAXN];
int n, m, c[MAXN][2], f[MAXN], p[MAXN], s[MAXN], flag[MAXN];
bool r[MAXN];
```

```
void init(int x, int value) {
    d[x] = g[x] = MsgNode(value);
    c[x][0] = c[x][1] = 0;
    f[x] = p[x] = flag[x] = -1;
    s[x] = 1;
}
void update(int x) {
    s[x] = s[c[x][0]] + s[c[x][1]] + 1;
    g[x] = MsgNode();
    if (c[x][0 ^ r[x]]) g[x] = g[x] + g[c[x][0 ^ r[x]]];
    g[x] = g[x] + d[x];
    if (c[x][1 ^ r[x]]) g[x] = g[x] + g[c[x][1 ^ r[x]]];
}
void makesame(int x, int c) {
    flag[x] = c;
    d[x] = MsgNode(c);
    g[x] = MsgNode(c);
}
void pushdown(int x) {
    if (r[x]) {
        std::swap(c[x][0], c[x][1]);
        r[c[x][0]] ^= 1;
        r[c[x][1]] ^= 1;
        std::swap(g[c[x][0]].leftColor, g[c[x][0]].rightColor);
        std::swap(g[c[x][1]].leftColor, g[c[x][1]].rightColor);
        r[x] = false;
    }
    if (flag[x] != -1) {
        if (c[x][0]) makesame(c[x][0], flag[x]);
        if (c[x][1]) makesame(c[x][1], flag[x]);
        flag[x] = -1;
    }
}
void rotate(int x, int k) {
    pushdown(x); pushdown(c[x][k]);
    int y = c[x][k]; c[x][k] = c[y][k ^ 1]; c[y][k ^ 1] = x;
    if (f[x] != -1) c[f[x]][c[f[x]][1] == x] = y;
    f[y] = f[x]; f[x] = y; f[c[x][k]] = x; std::swap(p[x], p[y]);
    update(x); update(y);
}
void splay(int x, int s = -1) {
    pushdown(x);
    while (f[x] != s) {
        if (f[f[x]] != s) rotate(f[f[x]], (c[f[f[x]]][1] == f[x]) ^ r[f[f[x]]]);
        rotate(f[x], (c[f[x]][1] == x) ^ r[f[x]]);
    }
    update(x);
}
void access(int x) {
    int y = 0;
    while (x != -1) {
        splay(x); pushdown(x);
        f[c[x][1]] = -1; p[c[x][1]] = x;
```

```
        c[x][1] = y; f[y] = x; p[y] = -1;
        update(x); x = p[y = x];
    }
}
void setroot(int x) {
    access(x);
    splay(x);
    r[x] ^= 1;
    std::swap(g[x].leftColor, g[x].rightColor);
}
void link(int x, int y) {
    setroot(x);
    p[x] = y;
}
```

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 点双连通分量

4.2.1 坚固的点双连通分量

```cpp
int n, m, x, y, ans1, ans2, tot1, tot2, flag, size, ind2, dfn[N], low[N], block[M], vis[N];
vector<int> a[N];
pair<int, int> stack[M];
void tarjan(int x, int p) {
    dfn[x] = low[x] = ++ind2;
    for (int i = 0; i < a[x].size(); ++i)
        if (dfn[x] > dfn[a[x][i]] && a[x][i] != p){
            stack[++size] = make_pair(x, a[x][i]);
            if (i == a[x].size() - 1 || a[x][i] != a[x][i + 1])
                if (!dfn[a[x][i]]){
                    tarjan(a[x][i], x);
                    low[x] = min(low[x], low[a[x][i]]);
                    if (low[a[x][i]] >= dfn[x]){
                        tot1 = tot2 = 0;
                        ++flag;
                        for (; ; ){
                            if (block[stack[size].first] != flag) {
                                ++tot1;
                                block[stack[size].first] = flag;
                            }
                            if (block[stack[size].second] != flag) {
                                ++tot1;
                                block[stack[size].second] = flag;
                            }
                            if (stack[size].first == x && stack[size].second == a[x][i])
                                break;
                            ++tot2;
                            --size;
                        }
                        for (; stack[size].first == x && stack[size].second == a[x][i]; --size
                            )
                            ++tot2;
                        if (tot2 < tot1)
                            ans1 += tot2;
                        if (tot2 > tot1)
                            ans2 += tot2;
                    }
                }
                else
                    low[x] = min(low[x], dfn[a[x][i]]);
        }
}
int main(){
    for (; ; ){
        scanf("%d%d", &n, &m);
        if (n == 0 && m == 0) return 0;
        for (int i = 1; i <= n; ++i) {
            a[i].clear();
            dfn[i] = 0;
```

```
        }
        for (int i = 1; i <= m; ++i){
            scanf("%d%d",&x, &y);
            ++x, ++y;
            a[x].push_back(y);
            a[y].push_back(x);
        }
        for (int i = 1; i <= n; ++i)
            sort(a[i].begin(), a[i].end());
        ans1 = ans2 = ind2 = 0;
        for (int i = 1; i <= n; ++i)
            if (!dfn[i]) {
                size = 0;
                tarjan(i, 0);
            }
        printf("%d␣%d\n", ans1, ans2);
    }
    return 0;
}
```

4.2.2 朴素的点双连通分量

```
void tarjan(int x){
    dfn[x] = low[x] = ++ind2;
    v[x] = 1;
    for (int i = nt[x]; pt[i]; i = nt[i])
        if (!dfn[pt[i]]){
            tarjan(pt[i]);
            low[x] = min(low[x], low[pt[i]]);
            if (dfn[x] <= low[pt[i]])
                ++v[x];
        }
        else
            low[x] = min(low[x], dfn[pt[i]]);
}
int main(){
    for (; ; ){
        scanf("%d%d", &n, &m);
        if (n == 0 && m == 0)
            return 0;
        for (int i = 1; i <= ind; ++i)
            nt[i] = pt[i] = 0;
        ind = n;
        for (int i = 1; i <= ind; ++i)
            last[i] = i;
        for (int i = 1; i <= m; ++i){
            scanf("%d%d", &x, &y);
            ++x, ++y;
            edge(x, y), edge(y, x);
        }
        memset(dfn, 0, sizeof(dfn));
        memset(v, 0, sizeof(v));
```

```
        ans = num = ind2 = 0;
        for (int i = 1; i <= n; ++i)
            if (!dfn[i]){
                root = i;
                size = 0;
                ++num;
                tarjan(i);
                --v[root];
            }
        for (int i = 1; i <= n; ++i)
            if (v[i] + num - 1 > ans)
                ans = v[i] + num - 1;
        printf("%d\n",ans);
    }
    return 0;
}
```

4.3 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;
}
```

4.4 二分图最大匹配

4.4.1 Hungary 算法

时间复杂度：$\mathcal{O}(V \cdot E)$

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

bool dfs(int x) {
    for (int i = 0; i < (int)edge[x].size(); ++i) {
        int y = edge[x][i];
        if (visit[y] != stamp) {
            visit[y] = stamp;
            if (match[y] == -1 || dfs(match[y])) {
                match[y] = x;
                return true;
            }
        }
    }
    return false;
}

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

4.4.2 Hopcroft Karp 算法

时间复杂度：$\mathcal{O}(\sqrt{V} \cdot E)$

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

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

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

4.5 二分图最大权匹配

时间复杂度：$\mathcal{O}(V^4)$

```
int DFS(int x){
    visx[x] = 1;
    for (int y = 1;y <= ny;y ++){
        if (visy[y]) continue;
        int t = lx[x] + ly[y] - w[x][y];
        if (t == 0) {
            visy[y] = 1;
            if (link[y] == -1||DFS(link[y])){
                link[y] = x;
                return 1;
            }
        }
        else slack[y] = min(slack[y],t);
    }
    return 0;
}
int KM(){
    int i,j;
    memset(link,-1,sizeof(link));
    memset(ly,0,sizeof(ly));
    for (i = 1; i <= nx; i++)
        for (j = 1, lx[i] = -inf; j <= ny; j++)
            lx[i] = max(lx[i],w[i][j]);
    for (int x = 1; x <= nx; x++){
        for (i = 1; i <= ny; i++) slack[i] = inf;
        while (true) {
            memset(visx, 0, sizeof(visx));
            memset(visy, 0, sizeof(visy));
            if (DFS(x)) break;
            int d = inf;
            for (i = 1; i <= ny;i++)
                if (!visy[i] && d > slack[i]) d = slack[i];
            for (i = 1; i <= nx; i++)
                if (visx[i]) lx[i] -= d;
            for (i = 1; i <= ny; i++)
                if (visy[i]) ly[i] += d;
                else slack[i] -= d;
        }
    }
    int res = 0;
    for (i = 1;i <= ny;i ++)
        if (link[i] > -1) res += w[link[i]][i];
    return res;
}
```

4.6 最大流

4.6.1 Dinic

使用方法以及注意事项：n 个点，m 条边，inf 为一个很大的值，源点 s，汇点 t，图中最大点的编号为 t。邻接表：p 数组记录节点，nxt 数组指向下一个位置,c 数组记录可增广量，h 数组记录表头 (初始全为-1)。时间复杂度：$\mathcal{O}(V^2 \cdot E)$

```
int bfs(){
    for (int i = 1;i <= t;i ++) d[i] = -1;
    int l,r;
    q[l = r = 0] = s, d[s] = 0;
    for (;l <= r;l ++)
        for (int k = h[q[l]]; k > -1; k = nxt[k])
            if (d[p[k]] == -1 && c[k] > 0) d[p[k]] = d[q[l]] + 1, q[++ r] = p[k];
    return d[t] > -1 ? 1 : 0;
}
int dfs(int u,int ext){
    if (u == t) return ext;
    int k = w[u],ret = 0;
    for (; k > -1; k = nxt[k], w[u] = k){      //w数组为当前弧
        if (ext == 0) break;
        if (d[p[k]] == d[u] + 1 && c[k] > 0){
            int flow = dfs(p[k], min(c[k], ext));
            if (flow > 0){
                c[k] -= flow, c[k ^ 1] += flow;
                ret += flow, ext -= flow;     //ret累计增广量，ext记录还可增广的量
            }
        }
    }
    if (k == -1) d[u] = -1;
    return ret;
}
void dinic() {
    while (bfs()) {
        for (int i = 1; i <= t;i ++) w[i] = h[i];
        dfs(s, inf);
    }
}
```

4.6.2 ISAP

时间复杂度：$\mathcal{O}(V^2 \cdot E)$

```
int Maxflow_Isap(int s,int t,int n) {
    std::fill(pre + 1, pre + n + 1, 0);
    std::fill(d + 1, d + n + 1, 0);
    std::fill(gap + 1, gap + n + 1, 0);
    for (int i = 1; i <= n; i++) cur[i] = h[i];
    gap[0] = n;
    int u = pre[s] = s, v, maxflow = 0;
    while (d[s] < n) {
        v = n + 1;
```

```
        for (int i = cur[u]; i; i = e[i].next)
        if (e[i].flow && d[u] == d[e[i].node] + 1) {
            v = e[i].node; cur[u]=i; break;
        }
        if (v <= n) {
            pre[v] = u; u = v;
            if (v == t) {
                int dflow = INF, p = t; u = s;
                while (p != s) {
                    p = pre[p];
                    dflow = std::min(dflow, e[cur[p]].flow);
                }
                maxflow += dflow; p = t;
                while (p != s) {
                    p = pre[p];
                    e[cur[p]].flow -= dflow;
                    e[e[cur[p]].opp].flow += dflow;
                }
            }
        }
        else{
            int mindist = n + 1;
            for (int i = h[u]; i; i = e[i].next)
                if (e[i].flow && mindist > d[e[i].node]) {
                    mindist = d[e[i].node]; cur[u] = i;
                }
            if (!--gap[d[u]]) return maxflow;
            gap[d[u] = mindist + 1]++; u = pre[u];
        }
    }
    return maxflow;
}
```

4.6.3 SAP

时间复杂度：$\mathcal{O}(V^2 \cdot E)$

```
const int N = 110, M = 30110, INF = 1000000000;//边表不要开小
int n, m, ind, S, T, flow, tot, pt[M], nt[M], last[N], size[M], num[N], h[N], now[N];
void edge(int x, int y, int z){
    last[x] = nt[last[x]] = ++ind;
    pt[ind] = y, size[ind] = z;
}
int aug(int x, int y){
    if (x == T)
        return y;
    int f = y;
    for (int i = now[x]; pt[i]; i = nt[i])
        if (size[i] && h[pt[i]] + 1 == h[x]){
            int z = aug(pt[i], min(f, size[i]));
            f -= z;
            size[i] -= z;
            size[i ^ 1] += z;
```

```
            now[x] = i;
            if (h[S] > tot || f == 0)
                return y - f;
        }
    now[x] = nt[x];
    if (--num[h[x]] == 0)
        h[S] = tot + 1;
    ++num[++h[x]];
    return y - f;
}
int main(){
    int np, nc;
    for (; scanf("%d%d%d%d", &n, &np, &nc, &m) == 4; ) {
        for (int i = 0; i <= ind; ++i)
            pt[i] = nt[i] = last[i] = size[i] = 0;
        ind = n + 2;
        if (ind % 2 == 0)
            ++ind;
        S = n + 1, tot = T = n + 2;
        for (int i = 0; i <= tot; ++i)
            num[i] = h[i] = now[i] = 0;
        for (int i = 1; i <= tot; ++i)
            last[i] = i;
        for (int i = 1; i <= m; ++i){
            int x, y, z;
            for (; getchar() != '('; );
            scanf("%d%*c%d%*c%d", &x, &y, &z);
            ++x, ++y;
            edge(x, y, z);
            edge(y, x, 0);
        }
        for (int i = 1; i <= np; ++i) {
            int y, z;
            for (; getchar() != '('; );
            scanf("%d%*c%d", &y, &z);
            ++y;
            edge(S, y, z);
            edge(y, S, 0);
        }
        for (int i = 1; i <= nc; ++i) {
            int x, z;
            for (; getchar() != '('; );
            scanf("%d%*c%d", &x, &z);
            ++x;
            edge(x, T, z);
            edge(T, x, 0);
        }
        num[0] = tot;
        for (int i = 1; i <= tot; ++i)
            now[i] = nt[i];
        flow = 0;
        for (; h[S] <= T; )
            flow += aug(S, INF);
```

```
        printf("%d\n", flow);
    }
    return 0;
}
```

4.7 上下界网络流

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

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

4.7.1 无源汇的上下界可行流

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

4.7.2 有源汇的上下界可行流

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

4.7.3 有源汇的上下界最大流

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

2. 从汇点 T 到源点 S 连一条上界为 ∞，下界为 0 的边，变成无源汇的网络。按照**无源汇的上下界可行流**的方法，建立超级源点 S^* 和超级汇点 T^*，求一遍 $S^* \to T^*$ 的最大流，再将从汇点 T 到源点 S 的这条边拆掉，求一次 $S \to T$ 的最大流即可。

4.7.4 有源汇的上下界最小流

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

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

4.8 最小费用最大流

4.8.1 稀疏图

时间复杂度：$\mathcal{O}(V \cdot E^2)$

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

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

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

bool augment() {
    static int dist[N], occur[N];
    std::vector<int> queue;
    std::fill(dist, dist + n, INT_MAX);
    std::fill(occur, occur + n, 0);
    dist[source] = 0;
    occur[source] = true;
    queue.push_back(source);
    for (int head = 0; head < (int)queue.size(); ++head) {
        int x = queue[head];
        for (int i = e.last[x]; ~i; i = e.succ[i]) {
            int y = e.other[i];
            if (e.flow[i] && dist[y] > dist[x] + e.cost[i]) {
                dist[y] = dist[x] + e.cost[i];
                prev[y] = i;
                if (!occur[y]) {
                    occur[y] = true;
                    queue.push_back(y);
                }
            }
        }
        occur[x] = false;
    }
    return dist[target] < INT_MAX;
}

std::pair<int, int> solve() {
    std::pair<int, int> answer = std::make_pair(0, 0);
```

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

4.8.2 稠密图

使用条件：费用非负
时间复杂度：$\mathcal{O}(V \cdot E^2)$

```
int aug(int no,int res) {
    if(no == t) return cost += pi1 * res,res;
    v[no] = true;
    int flow = 0;
    for(int i = h[no]; ~ i ;i = nxt[i])
        if(cap[i] && !expense[i] && !v[p[i]]) {
            int d = aug(p[i],min(res,cap[i]));
            cap[i] -= d,cap[i ^ 1] += d,flow += d,res -= d;
            if( !res ) return flow;
        }
    return flow;
}
bool modlabel() {
    int d = maxint;
    for(int i = 1;i <= t;++ i)
        if(v[i]) {
            for(int j = h[i]; ~ j ;j = nxt[j])
                if(cap[j] && !v[p[j]] && expense[j] < d) d = expense[j];
        }
    if(d == maxint)return false;
    for(int i = 1;i <= t;++ i)
        if(v[i]) {
            for(int j = h[i];~ j;j = nxt[j])
                expense[j] -= d,expense[j ^ 1] += d;
        }
    pi1 += d;
    return true;
}
void minimum_cost_flow_zkw() {
    cost = 0;
    do{
        do{
            memset(v,false,sizeof v);
```

```
        }while (aug(s,maxint));
    }while (modlabel());
}
```

4.9 一般图最大匹配

时间复杂度：$\mathcal{O}(V^3)$

```
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;
}
```

4.10 无向图全局最小割

时间复杂度：$\mathcal{O}(V^3)$
注意事项：处理重边时，应该对边权累加

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

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

```
                node[max] = node[--n];
            }
            visit[node[max]] = true;
            prev = max;
            max = -1;
            for (int j = 1; j < n; ++j) {
                if (!visit[node[j]]) {
                    dist[node[j]] += graph[node[prev]][node[j]];
                    if (max == -1 || dist[node[max]] < dist[node[j]]) {
                        max = j;
                    }
                }
            }
        }
    }
    return answer;
}
```

4.11 最小树形图

```
int n, m, used[N], pass[N], eg[N], more, queue[N];
double g[N][N];

void combine(int id, double &sum) {
    int tot = 0, from, i, j, k;
    for (; id != 0 && !pass[id]; id = eg[id]) {
        queue[tot++] = id;
        pass[id] = 1;
    }

    for (from = 0; from < tot && queue[from] != id; from++);
    if  (from == tot) return;
    more = 1;
    for (i = from; i < tot; i++) {
        sum += g[eg[queue[i]]][queue[i]];
        if (i != from) {
            used[queue[i]] = 1;
            for (j = 1; j <= n; j++) if (!used[j]) {
                if (g[queue[i]][j] < g[id][j]) g[id][j] = g[queue[i]][j];
            }
        }
    }

    for (i = 1; i <= n; i++) if (!used[i] && i != id) {
        for (j = from; j < tot; j++) {
            k = queue[j];
            if (g[i][id] > g[i][k] - g[eg[k]][k]) g[i][id] = g[i][k] - g[eg[k]][k];
        }
    }
}

```

```
double mdst(int root) {
    int i, j, k;
    double sum = 0;
    memset(used, 0, sizeof(used));
    for (more = 1; more; ) {
        more = 0;
        memset(eg, 0, sizeof(eg));
        for (i = 1; i <= n; i++) if (!used[i] && i != root) {
            for (j = 1, k = 0; j <= n; j++) if (!used[j] && i != j)
                if (k == 0 || g[j][i] < g[k][i]) k = j;
            eg[i] = k;
        }

        memset(pass, 0, sizeof(pass));
        for (i = 1; i <= n; i++) if (!used[i] && !pass[i] && i != root) combine(i, sum);
    }

    for (i = 1; i <= n; i++) if (!used[i] && i != root) sum += g[eg[i]][i];
    return sum;
}
```

4.12 有根树的同构

时间复杂度：$\mathcal{O}(VlogV)$

```
const unsigned long long MAGIC = 4423;

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

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

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

```
        }
        std::sort(value.begin(), value.end());

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

4.13 度限制生成树

```
int n, m, S, K, ans , cnt , Best[N], fa[N], FE[N];
int f[N], p[M], t[M], c[M], o, Cost[N];
bool u[M], d[M];
pair<int, int> MinCost[N];
struct Edge {
    int a, b, c;
    bool operator < (const Edge & E) const { return c < E.c; }
}E[M];
vector<int> SE;
inline int F(int x) {
    return fa[x] == x ? x : fa[x] = F(fa[x]);
}
inline void AddEdge(int a, int b, int C) {
    p[++o] = b; c[o] = C;
    t[o] = f[a]; f[a] = o;
}
void dfs(int i, int father) {
    fa[i] = father;
    if (father == S) Best[i] = -1;
    else {
        Best[i] = i;
        if (~Best[father] && Cost[Best[father]] > Cost[i]) Best[i] = Best[father];
    }
    for (int j = f[i]; j; j = t[j])
    if (!d[j] && p[j] != father) {
        Cost[p[j]] = c[j];
        FE[p[j]] = j;
        dfs(p[j], i);
    }
}
inline bool Kruskal() {
    cnt = n - 1,ans = 0; o = 1;
    for (int i = 1; i <= n; i++) fa[i] = i, f[i] = 0;
    sort(E + 1, E + m + 1);
    for (int i = 1; i <= m; i++) {
```

```
        if (E[i].b == S) swap(E[i].a, E[i].b);
        if (E[i].a != S && F(E[i].a) != F(E[i].b)) {
            fa[F(E[i].a)] = F(E[i].b);
            ans += E[i].c;
            cnt --;
            u[i] = true;
            AddEdge(E[i].a, E[i].b, E[i].c);
            AddEdge(E[i].b, E[i].a, E[i].c);
        }
    }
    for (int i = 1; i <= n; i++) MinCost[i] = make_pair(INF, INF);
    for (int i = 1; i <= m; i++)
    if (E[i].a == S) {
        SE.push_back(i);
        MinCost[F(E[i].b)] = min(MinCost[F(E[i].b)], make_pair(E[i].c, i));
    }
    int dif = 0;
    for (int i = 1; i <= n; i++)
    if (i != S && fa[i] == i) {
        if (MinCost[i].second == INF) return false;
        if (++ dif > K) return false;
        dfs(E[MinCost[i].second].b, S);
        u[MinCost[i].second] = true;
        ans += MinCost[i].first;
    }
    return true;
}
bool Solve() {
    memset(d,false,sizeof d);
    memset(u,false,sizeof u);
    if (!Kruskal()) return false;
    for (int i = cnt + 1; i <= K && i <= n; i++) {
        int MinD = INF, MinID = -1;
        for (int j = (int) SE.size() - 1; j >= 0; j--)
        if (u[SE[j]])
            SE.erase(SE.begin() + j);
        for (int j = 0; j < (int) SE.size(); j++) {
            int tmp = E[SE[j]].c - Cost[Best[E[SE[j]].b]];
            if (tmp < MinD) {
                MinD = tmp;
                MinID= SE[j];
            }
        }
        if (MinID == -1) return true;
        if (MinD >= 0) break;
        ans += MinD;
        u[MinID] = true;
        d[FE[Best[E[MinID].b]]] = d[FE[Best[E[MinID].b]] ^ 1] = true;
        dfs(E[MinID].b, S);
    }
    return true;
}
int main(){
```

```
    Solve();
    return 0;
}
```

4.14 弦图相关

4.14.1 弦图的判定

```
int n, m, first[1001], l, next[2000001], where[2000001],f[1001], a[1001], c[1001], L[1001], R
    [1001],
v[1001], idx[1001], pos[1001];
bool b[1001][1001];

inline void makelist(int x, int y){
    where[++l] = y;
    next[l] = first[x];
    first[x] = l;
}

bool cmp(const int &x, const int &y){
    return(idx[x] < idx[y]);
}

int main(){
    for (;;)
    {
        n = read(); m = read();
        if (!n && !m) return 0;
        memset(first, 0, sizeof(first)); l = 0;
        memset(b, false, sizeof(b));
        for (int i = 1; i <= m; i++)
        {
            int x = read(), y = read();
            if (x != y && !b[x][y])
            {
               b[x][y] = true; b[y][x] = true;
               makelist(x, y); makelist(y, x);
            }
        }
        memset(f, 0, sizeof(f));
        memset(L, 0, sizeof(L));
        memset(R, 255, sizeof(R));
        L[0] = 1; R[0] = n;
        for (int i = 1; i <= n; i++) c[i] = i, pos[i] = i;
        memset(idx, 0, sizeof(idx));
        memset(v, 0, sizeof(v));
        for (int i = n; i; --i)
        {
            int now = c[i];
            R[f[now]]--;
            if (R[f[now]] < L[f[now]]) R[f[now]] = -1;
```

```
            idx[now] = i; v[i] = now;
            for (int x = first[now]; x; x = next[x])
                if (!idx[where[x]])
                {
                   swap(c[pos[where[x]]], c[R[f[where[x]]]]);
                   pos[c[pos[where[x]]]] = pos[where[x]];
                   pos[where[x]] = R[f[where[x]]];
                   L[f[where[x]] + 1] = R[f[where[x]]]--;
                   if (R[f[where[x]]] < L[f[where[x]]]) R[f[where[x]]] = -1;
                   if (R[f[where[x]] + 1] == -1)
                       R[f[where[x]] + 1] = L[f[where[x]] + 1];
                   ++f[where[x]];
                }
        }
        bool ok = true;
        //v是完美消除序列.
        for (int i = 1; i <= n && ok; i++)
        {
            int cnt = 0;
            for (int x = first[v[i]]; x; x = next[x])
                if (idx[where[x]] > i) c[++cnt] = where[x];
            sort(c + 1, c + cnt + 1, cmp);
            bool can = true;
            for (int j = 2; j <= cnt; j++)
                if (!b[c[1]][c[j]])
                {
                    ok = false;
                    break;
                }
        }
        if (ok) printf("Perfect\n");
        else printf("Imperfect\n");
        printf("\n");
    }
}
```

4.14.2 弦图的团数

```
int n, m, first[100001], next[2000001], where[2000001], l, L[100001], R[100001], c[100001], f[100001],
pos[100001], idx[100001], v[100001], ans;

inline void makelist(int x, int y){
    where[++l] = y;
    next[l] = first[x];
    first[x] = l;
}

int read(){
    char ch;
    for (ch = getchar(); ch < '0' || ch > '9'; ch = getchar());
    int cnt = 0;
```

```
    for (; ch >= '0' && ch <= '9'; ch = getchar()) cnt = cnt * 10 + ch - '0';
    return(cnt);
}

int main(){
    //freopen("1006.in", "r", stdin);
    //freopen("1006.out", "w", stdout);
    memset(first, 0, sizeof(first)); l = 0;
    n = read(); m = read();
    for (int i = 1; i <= m; i++)
    {
        int x, y;
        x = read(); y = read();
        makelist(x, y); makelist(y, x);
    }
    memset(L, 0, sizeof(L));
    memset(R, 255, sizeof(R));
    memset(f, 0, sizeof(f));
    memset(idx, 0, sizeof(idx));
    for (int i = 1; i <= n; i++) c[i] = i, pos[i] = i;
    L[0] = 1; R[0] = n; ans = 0;
    for (int i = n; i; --i)
    {
        int now = c[i], cnt = 1;
        idx[now] = i; v[i] = now;
        if (--R[f[now]] < L[f[now]]) R[f[now]] = -1;
        for (int x = first[now]; x; x = next[x])
            if (!idx[where[x]])
            {
                swap(c[pos[where[x]]], c[R[f[where[x]]]]);
                pos[c[pos[where[x]]]] = pos[where[x]];
                pos[where[x]] = R[f[where[x]]];
                L[f[where[x]] + 1] = R[f[where[x]]]--;
                if (R[f[where[x]]] < L[f[where[x]]]) R[f[where[x]]] = -1;
                if (R[f[where[x]] + 1] == -1) R[f[where[x]] + 1] = L[f[where[x]] + 1];
                ++f[where[x]];
            }
            else ++cnt;
        ans = max(ans, cnt);
    }
    printf("%d\n", ans);
}
```

4.15 哈密尔顿回路（ORE 性质的图）

ORE 性质:

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

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

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

```cpp

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 坚固的模式串匹配

```
lenA = strlen(A); lenB = strlen(B);
nxt[0] = lenB,nxt[1] = lenB - 1;
for (int i = 0;i <= lenB;i ++)
    if (B[i] != B[i + 1]) {nxt[1] = i; break;}
int j, k = 1, p, L;
for (int i = 2;i < lenB;i ++) {
    p = k + nxt[k] - 1; L = nxt[i - k];
    if (i + L <= p) nxt[i] = L;
    else {
        j = p - i + 1;
        if (j < 0) j = 0;
        while (i + j < lenB && B[i + j] == B[j]) j++;
        nxt[i] = j; k = i;
```

```
    }
}
int minlen = lenA <= lenB ? lenA : lenB; ex[0] = minlen;
for (int i = 0;i < minlen;i ++)
    if (A[i] != B[i]) {ex[0] = i; break;}
k = 0;
for (int i = 1;i < lenA;i ++){
    p = k + ex[k] - 1; L = next[i - k];
    if (i + L <= p) ex[i] = L;
    else {
        j = p - i + 1;
        if (j < 0) j = 0;
        while (i + j < lenA && j < lenB && A[i + j] == B[j]) j++;
        ex[i] = j; k = i;
    }
}
```

5.3 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];
        }
}
```

5.4 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++);
    }
};
```

5.5 广义后缀自动机

```
// Generalized Suffix Automaton
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;
    }
}
```

5.6 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;
        }
    }
}
```

5.7 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 - l[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 - l[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];
        }
    }
}
```

5.8 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);
}
```

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

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

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

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

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

6.2 三维基础

6.2.1 点类

```cpp
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;
}
```

6.2.3 绕轴旋转

使用方法及注意事项：逆时针绕轴 AB 旋转 θ 角

```
Matrix getTrans(const double &a, const double &b, const double &c) {
    Matrix ret;
    ret.a[0][0] = 1; ret.a[0][1] = 0; ret.a[0][2] = 0; ret.a[0][3] = 0;
    ret.a[1][0] = 0; ret.a[1][1] = 1; ret.a[1][2] = 0; ret.a[1][3] = 0;
    ret.a[2][0] = 0; ret.a[2][1] = 0; ret.a[2][2] = 1; ret.a[2][3] = 0;
    ret.a[3][0] = a; ret.a[3][1] = b; ret.a[3][2] = c; ret.a[3][3] = 1;
    return ret;
```

```
}
Matrix getRotate(const double &a, const double &b, const double &c, const double &theta) {
    Matrix ret;
    ret.a[0][0] = a * a * (1 - cos(theta)) + cos(theta);
    ret.a[0][1] = a * b * (1 - cos(theta)) + c * sin(theta);
    ret.a[0][2] = a * c * (1 - cos(theta)) - b * sin(theta);
    ret.a[0][3] = 0;

    ret.a[1][0] = b * a * (1 - cos(theta)) - c * sin(theta);
    ret.a[1][1] = b * b * (1 - cos(theta)) + cos(theta);
    ret.a[1][2] = b * c * (1 - cos(theta)) + a * sin(theta);
    ret.a[1][3] = 0;

    ret.a[2][0] = c * a * (1 - cos(theta)) + b * sin(theta);
    ret.a[2][1] = c * b * (1 - cos(theta)) - a * sin(theta);
    ret.a[2][2] = c * c * (1 - cos(theta)) + cos(theta);
    ret.a[2][3] = 0;

    ret.a[3][0] = 0;
    ret.a[3][1] = 0;
    ret.a[3][2] = 0;
    ret.a[3][3] = 1;
    return ret;
}
Matrix getRotate(const double &ax, const double &ay, const double &az, const double &bx, const
    double &by, const double &bz, const double &theta) {
    double l = dist(Point(0, 0, 0), Point(bx, by, bz));
    Matrix ret = getTrans(-ax, -ay, -az);
    ret = ret * getRotate(bx / l, by / l, bz / l, theta);
    ret = ret * getTrans(ax, ay, az);
    return ret;
}
```

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 最小覆盖球

```
double eps(1e-8);
int sign(const double & x) {
    return (x > eps) - (x + eps < 0);
}
bool equal(const double & x, const double & y) {
    return x + eps > y and y + eps > x;
}
struct Point {
    double x, y, z;
    Point() {
    }
    Point(const double & x, const double & y, const double & z) : x(x), y(y), z(z){
    }
    void scan() {
        scanf("%lf%lf%lf", &x, &y, &z);
    }
    double sqrlen() const {
        return x * x + y * y + z * z;
    }
    double len() const {
        return sqrt(sqrlen());
    }
    void print() const {
        printf("(%lf %lf %lf)\n", x, y, z);
    }
} a[33];
Point operator + (const Point & a, const Point & b) {
    return Point(a.x + b.x, a.y + b.y, a.z + b.z);
}
Point operator - (const Point & a, const Point & b) {
    return Point(a.x - b.x, a.y - b.y, a.z - b.z);
}
Point operator * (const double & x, const Point & a) {
    return Point(x * a.x, x * a.y, x * a.z);
}
double operator % (const Point & a, const Point & b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
Point operator * (const Point & a, const Point & b) {
```

```
    return Point(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x);
}
struct Circle {
    double r;
    Point o;
    Circle() {
        o.x = o.y = o.z = r = 0;
    }
    Circle(const Point & o, const double & r) : o(o), r(r) {
    }
    void scan() {
        o.scan();
        scanf("%lf", &r);
    }
    void print() const {
        o.print();
        printf("%lf\n", r);
    }
};
struct Plane {
    Point nor;
    double m;
    Plane(const Point & nor, const Point & a) : nor(nor){
        m = nor % a;
    }
};
Point intersect(const Plane & a, const Plane & b, const Plane & c) {
    Point c1(a.nor.x, b.nor.x, c.nor.x), c2(a.nor.y, b.nor.y, c.nor.y), c3(a.nor.z, b.nor.z, c
        .nor.z), c4(a.m, b.m, c.m);
    return 1 / ((c1 * c2) % c3) * Point((c4 * c2) % c3, (c1 * c4) % c3, (c1 * c2) % c4);
}
bool in(const Point & a, const Circle & b) {
    return sign((a - b.o).len() - b.r) <= 0;
}
bool operator < (const Point & a, const Point & b) {
    if(!equal(a.x, b.x)) {
        return a.x < b.x;
    }
    if(!equal(a.y, b.y)) {
        return a.y < b.y;
    }
    if(!equal(a.z, b.z)) {
        return a.z < b.z;
    }
    return false;
}
bool operator == (const Point & a, const Point & b) {
    return equal(a.x, b.x) and equal(a.y, b.y) and equal(a.z, b.z);
}
vector<Point> vec;
Circle calc() {
    if(vec.empty()) {
        return Circle(Point(0, 0, 0), 0);
```

```
    }else if(1 == (int)vec.size()) {
        return Circle(vec[0], 0);
    }else if(2 == (int)vec.size()) {
        return Circle(0.5 * (vec[0] + vec[1]), 0.5 * (vec[0] - vec[1]).len());
    }else if(3 == (int)vec.size()) {
        double r((vec[0] - vec[1]).len() * (vec[1] - vec[2]).len() * (vec[2] - vec[0]).len() /
             2 / fabs(((vec[0] - vec[2]) * (vec[1] - vec[2])).len()));
        return Circle(intersect(Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0])),
                        Plane(vec[2] - vec[1], 0.5 * (vec[2] + vec[1])),
                    Plane((vec[1] - vec[0]) * (vec[2] - vec[0]), vec[0])), r);
    }else {
        Point o(intersect(Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0])),
                  Plane(vec[2] - vec[0], 0.5 * (vec[2] + vec[0])),
                  Plane(vec[3] - vec[0], 0.5 * (vec[3] + vec[0]))));
        return Circle(o, (o - vec[0]).len());
    }
}
Circle miniBall(int n) {
    Circle res(calc());
    for(int i(0); i < n; i++) {
        if(!in(a[i], res)) {
            vec.push_back(a[i]);
            res = miniBall(i);
            vec.pop_back();
            if(i) {
                Point tmp(a[i]);
                memmove(a + 1, a, sizeof(Point) * i);
                a[0] = tmp;
            }
        }
    }
    return res;
}
int main() {
    int n;
    for(;;) {
        scanf("%d", &n);
        if(!n) {
            break;
        }
        for(int i(0); i < n; i++) {
            a[i].scan();
        }
        sort(a, a + n);
        n = unique(a, a + n) - a;
        vec.clear();
        printf("%.10f\n", miniBall(n).r);
    }
```

6.4.3 多边形与圆的交面积

```
// 求扇形面积
```

```cpp
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。

7.2 博弈论相关

7.2.1 巴什博奕

1. 只有一堆 n 个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m 个。最后取光者得胜。

2. 显然，如果 $n = m + 1$，那么由于一次最多只能取 m 个，所以，无论先取者拿走多少个，后取者都能够一次拿走剩余的物品，后者取胜。因此我们发现了如何取胜的法则：如果 $n = \ m + 1\,r + s$，(r 为任意自然数，$s \leq m$), 那么先取者要拿走 s 个物品，如果后取者拿走 $k(k \leq m)$ 个，那么先取者再拿走 $m + 1 - k$ 个，结果剩下 $(m + 1)(r - 1)$ 个，以后保持这样的取法，那么先取者肯定获胜。总之，要保持给对手留下 $(m + 1)$ 的倍数，就能最后获胜。

7.2.2 威佐夫博弈

1. 有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

2. 判断一个局势 (a, b) 为奇异局势（必败态）的方法：

$$a_k = [k(1 + \sqrt{5})/2] \; b_k = a_k + k$$

7.2.3 阶梯博奕

1. 博弈在一列阶梯上进行，每个阶梯上放着自然数个点，两个人进行阶梯博弈，每一步则是将一个阶梯上的若干个点（至少一个）移到前面去，最后没有点可以移动的人输。
2. 解决方法：把所有奇数阶梯看成 N 堆石子，做 NIM。（把石子从奇数堆移动到偶数堆可以理解为拿走石子，就相当于几个奇数堆的石子在做 Nim）

7.2.4 图上删边游戏

链的删边游戏

1. 游戏规则：对于一条链，其中一个端点是根，两人轮流删边，脱离根的部分也算被删去，最后没边可删的人输。
2. 做法：$sg[i] = n - dist(i) - 1$（其中 n 表示总点数，$dist(i)$ 表示离根的距离）

树的删边游戏

1. 游戏规则：对于一棵有根树，两人轮流删边，脱离根的部分也算被删去，没边可删的人输。
2. 做法：叶子结点的 $sg = 0$，其他节点的 sg 等于儿子结点的 $sg + 1$ 的异或和。

局部连通图的删边游戏

1. 游戏规则：在一个局部连通图上，两人轮流删边，脱离根的部分也算被删去，没边可删的人输。局部连通图的构图规则是，在一棵基础树上加边得到，所有形成的环保证不共用边，且只与基础树有一个公共点。
2. 做法：去掉所有的偶环，将所有的奇环变为长度为 1 的链，然后做树的删边游戏。

7.3 Java Reference

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

public class Main {
    static int get(char c) {
        if (c <= '9')
            return c - '0';
        else if (c <= 'Z')
            return c - 'A' + 10;
        else
            return c - 'a' + 36;
    }
    static char get(int x) {
        if (x <= 9)
            return (char)(x + '0');
        else if (x <= 35)
            return (char)(x - 10 + 'A');
        else
            return (char)(x - 36 + 'a');
    }
    static BigInteger get(String s, BigInteger x) {
```

```
        BigInteger ans = BigInteger.valueOf(0), now = BigInteger.valueOf(1);
        for (int i = s.length() - 1; i >= 0; i--) {
            ans = ans.add(now.multiply(BigInteger.valueOf(get(s.charAt(i)))));
            now = now.multiply(x);
        }
        return ans;
    }
    public static void main(String [] args) {
        Scanner cin = new Scanner(new BufferedInputStream(System.in));
        for (; ; ) {
            BigInteger x = cin.nextBigInteger();
            if (x.compareTo(BigInteger.valueOf(0)) == 0)
                break;
            String s = cin.next(), t = cin.next(), r = "";
            BigInteger ans = get(s, x).mod(get(t, x));
            if (ans.compareTo(BigInteger.valueOf(0)) == 0)
                r = "0";
            for (; ans.compareTo(BigInteger.valueOf(0)) > 0;) {
                r = get(ans.mod(x).intValue()) + r;
                ans = ans.divide(x);
            }
            System.out.println(r);
        }
    }
}

// Arrays
int a[];
.fill(a[, int fromIndex, int toIndex],val); | .sort(a[, int fromIndex, int toIndex])
// String
String s;
.charAt(int i); | compareTo(String) | compareToIgnoreCase () | contains(String) |
length () | substring(int l, int len)
// BigInteger
.abs() | .add() | bitLength () | subtract () | divide () | remainder () | divideAndRemainder
    () | modPow(b, c) |
pow(int) | multiply () | compareTo () |
gcd() | intValue () | longValue () | isProbablePrime(int c) (1 - 1/2^c) |
nextProbablePrime () | shiftLeft(int) | valueOf ()
// BigDecimal
.ROUND_CEILING | ROUND_DOWN_FLOOR | ROUND_HALF_DOWN | ROUND_HALF_EVEN | ROUND_HALF_UP |
    ROUND_UP
.divide(BigDecimal b, int scale , int round_mode) | doubleValue () | movePointLeft(int) | pow(
    int) |
setScale(int scale , int round_mode) | stripTrailingZeros ()
// StringBuilder
StringBuilder sb = new StringBuilder ();
sb.append(elem) | out.println(sb)
```

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

8.1.3 错排公式

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

8.1.4 莫比乌斯函数

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

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

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

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

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

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

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

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

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

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

有

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

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

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

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

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

6. 内切圆半径
$$r = \frac{S}{p} = \frac{arcsin\frac{B}{2} \cdot sin\frac{C}{2}}{sin\frac{B+C}{2}} = 4R \cdot sin\frac{A}{2} sin\frac{B}{2} sin\frac{C}{2}$$
$$= \sqrt{\frac{(p-a)(p-b)(p-c)}{p}} = p \cdot tan\frac{A}{2} tan\frac{B}{2} tan\frac{C}{2}$$

7. 外接圆半径
$$R = \frac{abc}{4S} = \frac{a}{2sinA} = \frac{b}{2sinB} = \frac{c}{2sinC}$$

8.2.2 四边形

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

1. $a^2+b^2+c^2+d^2 = D_1^2 + D_2^2 + 4M^2$
2. $S = \frac{1}{2} D_1 D_2 sinA$
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
$$S = \frac{nar}{2} = nr^2 \cdot tan\frac{A}{2} = \frac{nR^2}{2} \cdot sinA = \frac{na^2}{4 \cdot tan\frac{A}{2}}$$

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
$$S_1 = \frac{rl}{2} = \frac{r^2A}{2}$$
5. 弓形面积
$$S_2 = \frac{rl - a(r-h)}{2} = \frac{r^2}{2}(A - sinA)$$

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
$$S = 2\pi rh$$
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

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

正弦定理

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

三角形面积是 $A + B + C - \pi$

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

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