

Standard Code Library

Shanghai Jiao Tong University

October, 2015

Contents

| | | |
|----------|------------------|-----------|
| 1 | 数论算法 | 5 |
| 1.1 | 快速数论变换 | 5 |
| 1.2 | 多项式求逆 | 6 |
| 1.3 | 中国剩余定理 | 6 |
| 1.4 | Miller Rabin | 6 |
| 1.5 | Pollard Rho | 7 |
| 1.6 | 坚固的逆元 | 8 |
| 1.7 | 直线下整点个数 | 8 |
| 2 | 数值算法 | 9 |
| 2.1 | 快速傅立叶变换 | 9 |
| 2.2 | 单纯形法求解线性规划 | 9 |
| 2.3 | 自适应辛普森 | 11 |
| 3 | 数据结构 | 12 |
| 3.1 | Splay 普通操作版 | 12 |
| 3.2 | Splay 区间操作版 | 14 |
| 3.3 | 坚固的 Treap | 18 |
| 3.4 | k-d 树 | 20 |
| 3.5 | 树链剖分 | 21 |
| 3.5.1 | 点操作版本 | 21 |
| 3.5.2 | 链操作版本 | 25 |
| 3.6 | Link-Cut-Tree | 25 |
| 4 | 图论 | 28 |
| 4.1 | 强连通分量 | 28 |
| 4.2 | 点双连通分量 | 29 |
| 4.2.1 | 坚固的点双连通分量 | 29 |
| 4.2.2 | 朴素的点双连通分量 | 30 |
| 4.3 | 2-SAT 问题 | 31 |
| 4.4 | 二分图最大匹配 | 32 |
| 4.4.1 | Hungary 算法 | 32 |
| 4.4.2 | Hopcroft Karp 算法 | 32 |
| 4.5 | 二分图最大权匹配 | 34 |
| 4.6 | 最大流 | 35 |
| 4.6.1 | Dinic | 35 |
| 4.6.2 | ISAP | 35 |
| 4.6.3 | SAP | 36 |

| | | |
|----------|-------------------|-----------|
| 4.7 | 上下界网络流 | 38 |
| 4.7.1 | 无源汇的上下界可行流 | 38 |
| 4.7.2 | 有源汇的上下界可行流 | 38 |
| 4.7.3 | 有源汇的上下界最大流 | 38 |
| 4.7.4 | 有源汇的上下界最小流 | 38 |
| 4.8 | 最小费用最大流 | 38 |
| 4.8.1 | 稀疏图 | 38 |
| 4.8.2 | 稠密图 | 40 |
| 4.9 | 一般图最大匹配 | 41 |
| 4.10 | 无向图全局最小割 | 43 |
| 4.11 | 最小树形图 | 44 |
| 4.12 | 有根树的同构 | 45 |
| 4.13 | 度限制生成树 | 46 |
| 4.14 | 弦图相关 | 48 |
| 4.14.1 | 弦图的判定 | 48 |
| 4.14.2 | 弦图的团数 | 49 |
| 4.15 | 哈密尔顿回路 (ORE 性质的图) | 50 |
| 5 | 字符串 | 53 |
| 5.1 | 模式串匹配 | 53 |
| 5.2 | 坚固的模式串匹配 | 53 |
| 5.3 | AC 自动机 | 54 |
| 5.4 | 后缀数组 | 55 |
| 5.5 | 广义后缀自动机 | 56 |
| 5.6 | Manacher 算法 | 56 |
| 5.7 | 回文树 | 57 |
| 5.8 | 循环串最小表示 | 58 |
| 6 | 计算几何 | 59 |
| 6.1 | 二维基础 | 59 |
| 6.1.1 | 点类 | 59 |
| 6.1.2 | 凸包 | 59 |
| 6.1.3 | 半平面交 | 60 |
| 6.1.4 | 最近点对 | 60 |
| 6.2 | 三维基础 | 61 |
| 6.2.1 | 点类 | 61 |
| 6.2.2 | 凸包 | 63 |
| 6.2.3 | 绕轴旋转 | 65 |
| 6.3 | 多边形 | 66 |
| 6.3.1 | 判断点在多边形内部 | 66 |
| 6.3.2 | 多边形内整点计数 | 67 |
| 6.4 | 圆 | 67 |
| 6.4.1 | 最小覆盖圆 | 67 |
| 6.4.2 | 最小覆盖球 | 68 |
| 6.4.3 | 多边形与圆的交面积 | 70 |

| | | |
|----------|------------------|-----------|
| 7 | 其它 | 73 |
| 7.1 | STL 使用方法 | 73 |
| 7.1.1 | nth_element | 73 |
| 7.1.2 | next_permutation | 73 |
| 7.2 | 博弈论相关 | 73 |
| 7.2.1 | 巴什博奕 | 73 |
| 7.2.2 | 威佐夫博弈 | 73 |
| 7.2.3 | 阶梯博奕 | 74 |
| 7.2.4 | 图上删边游戏 | 74 |
| 7.3 | Java Reference | 74 |
| 8 | 数学公式 | 76 |
| 8.1 | 常用数学公式 | 76 |
| 8.1.1 | 求和公式 | 76 |
| 8.1.2 | 斐波那契数列 | 76 |
| 8.1.3 | 错排公式 | 76 |
| 8.1.4 | 莫比乌斯函数 | 77 |
| 8.1.5 | Burnside 引理 | 77 |
| 8.1.6 | 五边形数定理 | 77 |
| 8.1.7 | 树的计数 | 77 |
| 8.1.8 | 欧拉公式 | 78 |
| 8.1.9 | 皮克定理 | 78 |
| 8.1.10 | 牛顿恒等式 | 78 |
| 8.2 | 平面几何公式 | 79 |
| 8.2.1 | 三角形 | 79 |
| 8.2.2 | 四边形 | 79 |
| 8.2.3 | 正 n 边形 | 79 |
| 8.2.4 | 圆 | 80 |
| 8.2.5 | 棱柱 | 80 |
| 8.2.6 | 棱锥 | 81 |
| 8.2.7 | 棱台 | 81 |
| 8.2.8 | 圆柱 | 81 |
| 8.2.9 | 圆锥 | 81 |
| 8.2.10 | 圆台 | 82 |
| 8.2.11 | 球 | 82 |
| 8.2.12 | 球台 | 82 |
| 8.2.13 | 球扇形 | 82 |
| 8.3 | 立体几何公式 | 82 |
| 8.3.1 | 球面三角公式 | 82 |
| 8.3.2 | 四面体体积公式 | 83 |

Chapter 1

数论算法

1.1 快速数论变换

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

```
1  const int mod = 998244353;
2  const int pri = 3;
3  int prepare(int n) {
4      int len = 1;
5      for (; len <= 2 * n; len <<= 1);
6      for (int i = 0; i <= len; i++) {
7          e[0][i] = fpm(pri, (mod - 1) / len * i, mod);
8          e[1][i] = fpm(pri, (mod - 1) / len * (len - i), mod);
9      }
10     return len;
11 }
12 void DFT(int *a, int n, int f) {
13     for (int i = 0, j = 0; i < n; i++) {
14         if (i > j) std::swap(a[i], a[j]);
15         for (int t = n >> 1; (j ^= t) < t; t >>= 1);
16     }
17     for (int i = 2; i <= n; i <<= 1)
18         for (int j = 0; j < n; j += i)
19             for (int k = 0; k < (i >> 1); k++) {
20                 int A = a[j + k];
21                 int B = (long long)a[j + k + (i >> 1)] * e[f][n / i * k] % mod;
22                 a[j + k] = (A + B) % mod;
23                 a[j + k + (i >> 1)] = (A - B + mod) % mod;
24             }
25     if (f == 1) {
26         long long rev = fpm(n, mod - 2, mod);
27         for (int i = 0; i < n; i++) {
28             a[i] = (long long)a[i] * rev % mod;
29         }
30     }
31 }
```

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。

```

1 void getInv(int *a, int *b, int n) {
2     static int tmp[MAXN];
3     std::fill(b, b + n, 0);
4     b[0] = fpm(a[0], mod - 2, mod);
5     for (int c = 1; c <= n; c <<= 1) {
6         for (int i = 0; i < c; i++) tmp[i] = a[i];
7         std::fill(b + c, b + (c << 1), 0);
8         std::fill(tmp + c, tmp + (c << 1), 0);
9         DFT(tmp, c << 1, 0);
10        DFT(b, c << 1, 0);
11        for (int i = 0; i < (c << 1); i++) {
12            b[i] = (long long)(2 - (long long)tmp[i] * b[i] % mod + mod) * b[i] % mod;
13        }
14        DFT(b, c << 1, 1);
15        std::fill(b + c, b + (c << 1), 0);
16    }
17 }
```

1.3 中国剩余定理

使用条件及注意事项：模数可以不互质。

```

1 bool solve(int n, std::pair<long long, long long> input[],
2             std::pair<long long, long long> &output) {
3     output = std::make_pair(1, 1);
4     for (int i = 0; i < n; ++i) {
5         long long number, useless;
6         euclid(output.second, input[i].second, number, useless);
7         long long divisor = std::__gcd(output.second, input[i].second);
8         if ((input[i].first - output.first) % divisor) {
9             return false;
10        }
11        number *= (input[i].first - output.first) / divisor;
12        fix(number, input[i].second);
13        output.first += output.second * number;
14        output.second *= input[i].second / divisor;
15        fix(output.first, output.second);
16    }
17    return true;
18 }
```

1.4 Miller Rabin

```

1 const int BASE[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
2
3 bool check(const long long &prime, const long long &base) {
```

```

4     long long number = prime - 1;
5     for (; ~number & 1; number >>= 1);
6     long long result = power_mod(base, number, prime);
7     for (; number != prime - 1 && result != 1 && result != prime - 1; number <<= 1) {
8         result = multiply_mod(result, result, prime);
9     }
10    return result == prime - 1 || (number & 1) == 1;
11 }
12
13 bool miller_rabin(const long long &number) {
14     if (number < 2) {
15         return false;
16     }
17     if (number < 4) {
18         return true;
19     }
20     if (~number & 1) {
21         return false;
22     }
23     for (int i = 0; i < 12 && BASE[i] < number; ++i) {
24         if (!check(number, BASE[i])) {
25             return false;
26         }
27     }
28     return true;
29 }

```

1.5 Pollard Rho

```

1  long long pollard_rho(const long long &number, const long long &seed) {
2      long long x = rand() % (number - 1) + 1, y = x;
3      for (int head = 1, tail = 2; ; ) {
4          x = multiply_mod(x, x, number);
5          x = add_mod(x, seed, number);
6          if (x == y) {
7              return number;
8          }
9          long long answer = std::__gcd(abs(x - y), number);
10         if (answer > 1 && answer < number) {
11             return answer;
12         }
13         if (++head == tail) {
14             y = x;
15             tail <<= 1;
16         }
17     }
18 }
19
20 void factorize(const long long &number, std::vector<long long> &divisor) {
21     if (number > 1) {
22         if (miller_rabin(number)) {

```

```
23         divisor.push_back(number);
24     } else {
25         long long factor = number;
26         for (; factor >= number;
27             factor = pollard_rho(number, rand() % (number - 1) + 1));
28             factorize(number / factor, divisor);
29             factorize(factor, divisor);
30     }
31 }
32 }
```

1.6 坚固的逆元

```
1 long long inverse(const long long &x, const long long &mod) {
2     if (x == 1) {
3         return 1;
4     } else {
5         return (mod - mod / x) * inverse(mod % x, mod) % mod;
6     }
7 }
```

1.7 直线下整点个数

```
1 long long solve(const long long &n, const long long &a,
2                 const long long &b, const long long &m) {
3     if (b == 0) {
4         return n * (a / m);
5     }
6     if (a >= m) {
7         return n * (a / m) + solve(n, a % m, b, m);
8     }
9     if (b >= m) {
10        return (n - 1) * n / 2 * (b / m) + solve(n, a, b % m, m);
11    }
12    return solve((a + b * n) / m, (a + b * n) % m, m, b);
13 }
```


Chapter 2

数值算法

2.1 快速傅立叶变换

```
1  int prepare(int n) {
2      int len = 1;
3      for (; len <= 2 * n; len <<= 1);
4      for (int i = 0; i < len; i++) {
5          e[0][i] = Complex(cos(2 * pi * i / len), sin(2 * pi * i / len));
6          e[1][i] = Complex(cos(2 * pi * i / len), -sin(2 * pi * i / len));
7      }
8      return len;
9  }
10
11 void DFT(Complex *a, int n, int f) {
12     for (int i = 0, j = 0; i < n; i++) {
13         if (i > j) std::swap(a[i], a[j]);
14         for (int t = n >> 1; (j ^= t) < t; t >>= 1);
15     }
16     for (int i = 2; i <= n; i <<= 1)
17         for (int j = 0; j < n; j += i)
18             for (int k = 0; k < (i >> 1); k++) {
19                 Complex A = a[j + k];
20                 Complex B = e[f][n / i * k] * a[j + k + (i >> 1)];
21                 a[j + k] = A + B;
22                 a[j + k + (i >> 1)] = A - B;
23             }
24     if (f == 1) {
25         for (int i = 0; i < n; i++)
26             a[i].a /= n;
27     }
28 }
```

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 $\max\{c_{1 \times m} \cdot x_{m \times 1} \mid x_{m \times 1} \geq 0_{m \times 1}, a_{n \times m} \cdot x_{m \times 1} \leq b_{n \times 1}\}$

```

1  std::vector<double> solve(const std::vector<std::vector<double> > &a,
2                          const std::vector<double> &b, const std::vector<double> &c) {
3      int n = (int)a.size(), m = (int)a[0].size() + 1;
4      std::vector<std::vector<double> > value(n + 2, std::vector<double>(m + 1));
5      std::vector<int> index(n + m);
6      int r = n, s = m - 1;
7      for (int i = 0; i < n + m; ++i) {
8          index[i] = i;
9      }
10     for (int i = 0; i < n; ++i) {
11         for (int j = 0; j < m - 1; ++j) {
12             value[i][j] = -a[i][j];
13         }
14         value[i][m - 1] = 1;
15         value[i][m] = b[i];
16         if (value[r][m] > value[i][m]) {
17             r = i;
18         }
19     }
20     for (int j = 0; j < m - 1; ++j) {
21         value[n][j] = c[j];
22     }
23     value[n + 1][m - 1] = -1;
24     for (double number; ; ) {
25         if (r < n) {
26             std::swap(index[s], index[r + m]);
27             value[r][s] = 1 / value[r][s];
28             for (int j = 0; j <= m; ++j) {
29                 if (j != s) {
30                     value[r][j] *= -value[r][s];
31                 }
32             }
33             for (int i = 0; i <= n + 1; ++i) {
34                 if (i != r) {
35                     for (int j = 0; j <= m; ++j) {
36                         if (j != s) {
37                             value[i][j] += value[r][j] * value[i][s];
38                         }
39                     }
40                     value[i][s] *= value[r][s];
41                 }
42             }
43         }
44         r = s = -1;
45         for (int j = 0; j < m; ++j) {
46             if (s < 0 || index[s] > index[j]) {
47                 if (value[n + 1][j] > eps || value[n + 1][j] > -eps && value[n][j] > eps) {
48                     s = j;
49                 }
50             }
51         }
52         if (s < 0) {
53             break;

```

```

54     }
55     for (int i = 0; i < n; ++i) {
56         if (value[i][s] < -eps) {
57             if (r < 0
58                 || (number = value[r][m] / value[r][s] - value[i][m] / value[i][s]) < -eps
59                 || number < eps && index[r + m] > index[i + m]) {
60                 r = i;
61             }
62         }
63     }
64     if (r < 0) {
65         // Solution is unbounded.
66         return std::vector<double>();
67     }
68 }
69 if (value[n + 1][m] < -eps) {
70     // No solution.
71     return std::vector<double>();
72 }
73 std::vector<double> answer(m - 1);
74 for (int i = m; i < n + m; ++i) {
75     if (index[i] < m - 1) {
76         answer[index[i]] = value[i - m][m];
77     }
78 }
79 return answer;
80 }

```

2.3 自适应辛普森

```

1 double area(const double &left, const double &right) {
2     double mid = (left + right) / 2;
3     return (right - left) * (calc(left) + 4 * calc(mid) + calc(right)) / 6;
4 }
5
6 double simpson(const double &left, const double &right,
7               const double &eps, const double &area_sum) {
8     double mid = (left + right) / 2;
9     double area_left = area(left, mid);
10    double area_right = area(mid, right);
11    double area_total = area_left + area_right;
12    if (std::abs(area_total - area_sum) < 15 * eps) {
13        return area_total + (area_total - area_sum) / 15;
14    }
15    return simpson(left, mid, eps / 2, area_left)
16           + simpson(mid, right, eps / 2, area_right);
17 }
18
19 double simpson(const double &left, const double &right, const double &eps) {
20     return simpson(left, right, eps, area(left, right));
21 }

```

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)

```
1  int pred(int x) {
2      splay(x, -1);
3      for (x = c[x][0]; c[x][1]; x = c[x][1]);
4      return x;
5  }
6  int succ(int x) {
7      splay(x, -1);
8      for (x = c[x][1]; c[x][0]; x = c[x][0]);
9      return x;
10 }
11 void remove(int x) {
12     if (b[x] > 1) {b[x]--; splay(x, -1); return;}
13     splay(x, -1);
14     if (!c[x][0] && !c[x][1]) rt = 0;
15     else if (c[x][0] && !c[x][1]) f[rt = c[x][0]] = -1;
16     else if (!c[x][0] && c[x][1]) f[rt = c[x][1]] = -1;
17     else{
18         int t = pred(x); f[rt = c[x][0]] = -1;
19         c[t][1] = c[x][1]; f[c[x][1]] = t;
20         splay(c[x][1], -1);
21     }
```

```

22     c[x][0] = c[x][1] = f[x] = d[x] = s[x] = b[x] = 0;
23 }
24 int find(int z) {
25     int x=rt;
26     while (d[x]!=z)
27         if (c[x][d[x]<z]) x=c[x][d[x]<z];
28         else break;
29     return x;
30 }
31 void insert(int z) {
32     if (!rt) {
33         f[rt = ++size] = -1;
34         d[size] = z; b[size] = 1;
35         splay(size, -1);
36         return;
37     }
38     int x = find(z);
39     if (d[x] == z) {
40         b[x]++;
41         splay(x, -1);
42         return;
43     }
44     c[x][d[x]<z] = ++size; f[size] = x;
45     d[size] = z; b[size] = s[size] = 1;
46     splay(size, -1);
47 }
48 int select(int z) {
49     int x = rt;
50     while (z < s[c[x][0]] + 1 || z > s[c[x][0]] + b[x])
51         if (z > s[c[x][0]] + b[x]) {
52             z -= s[c[x][0]] + b[x];
53             x = c[x][1];
54         }
55     else x = c[x][0];
56     return x;
57 }
58 int main() {
59     scanf("%d",&n);
60     for (int i = 1; i <= n; i++) {
61         int opt, x;
62         scanf("%d%d", &opt, &x);
63         if (opt == 1) insert(x);
64         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)
65         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)
66             insert(x);
67             printf("%d\n", s[c[find(x)][0]] + 1);
68             remove(find(x));
69         }
70         else if (opt == 4) printf("%d\n",d[select(x)]);
71         else if (opt == 5) {
72             insert(x);
73             printf("%d\n", d[pred(find(x))]);
74             remove(find(x));

```

```

75     }
76     else if (opt == 6) {
77         insert(x);
78         printf("%d\n", d[succ(find(x))]);
79         remove(find(x));
80     }
81 }
82 return 0;
83 }

```

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。

```

1  const int INF = 1000000000;
2  const int Maxspace = 500000;
3  struct SplayNode{
4      int ls, rs, zs, ms;
5      SplayNode() {
6          ms = 0;
7          ls = rs = zs = -INF;
8      }
9      SplayNode(int d) {
10         ms = zs = ls = rs = d;
11     }
12     SplayNode operator +(const SplayNode &p) const {
13         SplayNode ret;
14         ret.ls = max(ls, ms + p.ls);
15         ret.rs = max(rs + p.ms, p.rs);
16         ret.zs = max(rs + p.ls, max(zs, p.zs));
17         ret.ms = ms + p.ms;
18         return ret;
19     }
20 }t[MAXN], d[MAXN];
21 int n, m, rt, top, a[MAXN], f[MAXN], c[MAXN][2], g[MAXN], h[MAXN], z[MAXN];
22 bool r[MAXN], b[MAXN];
23 void makesame(int x, int s) {
24     if (!x) return;
25     b[x] = true;
26     d[x] = SplayNode(g[x] = s);
27     t[x].zs = t[x].ms = g[x] * h[x];
28     t[x].ls = t[x].rs = max(g[x], g[x] * h[x]);
29 }
30 void makerev(int x) {
31     if (!x) return;
32     r[x] ^= 1;
33     swap(c[x][0], c[x][1]);
34     swap(t[x].ls, t[x].rs);
35 }
36 void pushdown(int x) {
37     if (!x) return;

```

```

38     if (r[x]) {
39         makerev(c[x][0]);
40         makerev(c[x][1]);
41         r[x]=0;
42     }
43     if (b[x]) {
44         makesame(c[x][0],g[x]);
45         makesame(c[x][1],g[x]);
46         b[x]=g[x]=0;
47     }
48 }
49 void updata(int x) {
50     if (!x) return;
51     h[x]=h[c[x][0]]+h[c[x][1]]+1;
52     t[x]=t[c[x][0]]+d[x]+t[c[x][1]];
53 }
54 void rotate(int x,int k) {
55     pushdown(x);pushdown(c[x][k]);
56     int y = c[x][k]; c[x][k] = c[y][k^1]; c[y][k^1] = x;
57     if (f[x] != -1) c[f[x]][c[f[x]][1] == x] = y; else rt = y;
58     f[y] = f[x]; f[x] = y; f[c[x][k]] = x;
59     updata(x); updata(y);
60 }
61 void splay(int x, int s) {
62     while (f[x] != s) {
63         if (f[f[x]]!=s) {
64             pushdown(f[f[x]]);
65             rotate(f[f[x]], (c[f[f[x]]][1] == f[x]) ^ r[f[f[x]]]);
66         }
67         pushdown(f[x]);
68         rotate(f[x], (c[f[x]][1]==x) ^ r[f[x]]);
69     }
70 }
71 void build(int &x,int l,int r) {
72     if (l > r) {x = 0; return;}
73     x = z[top--];
74     if (l < r) {
75         build(c[x][0],l,(l+r>>1)-1);
76         build(c[x][1],(l+r>>1)+1,r);
77     }
78     f[c[x][0]] = f[c[x][1]] = x;
79     d[x] = SplayNode(a[l+r>>1]);
80     updata(x);
81 }
82 void init() {
83     d[0] = SplayNode();
84     f[rt=2] = -1;
85     f[1] = 2; c[2][0] = 1;
86     int x;
87     build(x,1,n);
88     c[1][1] = x; f[x] = 1;
89     splay(x, -1);
90 }

```

```

91 int find(int z) {
92     int x = rt; pushdown(x);
93     while (z != h[c[x][0]] + 1) {
94         if (z > h[c[x][0]] + 1) {
95             z -= h[c[x][0]] + 1;
96             x = c[x][1];
97         }
98         else x = c[x][0];
99         pushdown(x);
100     }
101     return x;
102 }
103 void getrange(int &x,int &y) {
104     y = x + y - 1;
105     x = find(x);
106     y = find(y + 2);
107     splay(y, -1);
108     splay(x, y);
109 }
110 void recycle(int x) {
111     if (!x) return;
112     recycle(c[x][0]);
113     recycle(c[x][1]);
114     z[++top]=x;
115     t[x] = d[x] = SplayNode();
116     r[x] = b[x] = g[x] = f[x] = h[x] = 0;
117     c[x][0] = c[x][1]=0;
118 }
119 int main() {
120     scanf("%d%d",&n,&m);
121     for (int i = 1; i <= n; i++) scanf("%d",a+i);
122     for (int i = Maxspace; i>=3; i--) z[++top] = i;
123     init();
124     for (int i = 1; i <= m; i++) {
125         char op[10];
126         int x, y, tmp;
127         scanf("%s", op);
128         if (!strcmp(op, "INSERT")) {
129             scanf("%d%d", &x, &y);
130             n += y;
131             if (!y) continue;
132             for (int i = 1; i <= y; i++) scanf("%d",a+i);
133             build(tmp, 1, y);
134             x = find(x + 1); pushdown(x);
135             if (!c[x][1]) {c[x][1] = tmp; f[tmp] = x;}
136             else{
137                 x = c[x][1]; pushdown(x);
138                 while (c[x][0]) {
139                     x = c[x][0];
140                     pushdown(x);
141                 }
142                 c[x][0] = tmp; f[tmp] = x;
143             }

```



```

144         splay(tmp, -1);
145     }
146     else if (!strcmp(op, "DELETE")) {
147         scanf("%d%d", &x, &y); n -= y;
148         if (!y) continue;
149         getrange(x, y);
150         int k = (c[y][0] == x);
151         recycle(c[x][k]);
152         f[c[x][k]] = 0;
153         c[x][k] = 0;
154         splay(x, -1);
155     }
156     else if (!strcmp(op, "REVERSE")) {
157         scanf("%d%d", &x, &y);
158         if (!y) continue;
159         getrange(x, y);
160         int k = (c[y][0] == x);
161         makerev(c[x][k]);
162         splay(c[x][k], -1);
163     }
164     else if (!strcmp(op, "GET-SUM")) {
165         scanf("%d%d", &x, &y);
166         if (!y) {
167             printf("0\n");
168             continue;
169         }
170         getrange(x, y);
171         int k = (c[y][0] == x);
172         printf("%d\n", t[c[x][k]].ms);
173         splay(c[x][k], -1);
174     }
175     else if (!strcmp(op, "MAX-SUM")) {
176         x = 1; y = n;
177         getrange(x, y);
178         int k = (c[y][0] == x);
179         printf("%d\n", t[c[x][k]].zs);
180         splay(c[x][k], -1);
181     }
182     else if (!strcmp(op, "MAKE-SAME")) {
183         scanf("%d%d%d", &x, &y, &tmp);
184         if (!y) continue;
185         getrange(x, y);
186         int k = (c[y][0] == x);
187         makesame(c[x][k], tmp);
188         splay(c[x][k], -1);
189     }
190 }
191 return 0;
192 }

```

3.3 坚固的 Treap

使用条件及注意事项：题目来源 UVA 12358

```

1  int ran() {
2      static int ret = 182381727;
3      return (ret += (ret << 1) + 717271723) & (~0u >> 1);
4  }
5
6  int alloc(int node = 0) {
7      size++;
8      if (node) {
9          c[size][0] = c[node][0];
10         c[size][1] = c[node][1];
11         s[size] = s[node];
12         d[size] = d[node];
13     }
14     else{
15         c[size][0] = 0;
16         c[size][1] = 0;
17         s[size] = 1;
18         d[size] = '␣';
19     }
20     return size;
21 }
22
23 void update(int x) {
24     s[x] = 1;
25     if (c[x][0]) s[x] += s[c[x][0]];
26     if (c[x][1]) s[x] += s[c[x][1]];
27 }
28
29 int merge(const std::pair<int, int> &a) {
30     if (!a.first) return a.second;
31     if (!a.second) return a.first;
32     if (ran() % (s[a.first] + s[a.second]) < s[a.first]) {
33         int newnode = alloc(a.first);
34         c[newnode][1] = merge(std::make_pair(c[newnode][1], a.second));
35         update(newnode);
36         return newnode;
37     }
38     else{
39         int newnode = alloc(a.second);
40         c[newnode][0] = merge(std::make_pair(a.first, c[newnode][0]));
41         update(newnode);
42         return newnode;
43     }
44 }
45
46 std::pair<int, int> split(int x, int k) {
47     if (!x || !k) return std::make_pair(0, x);
48     int newnode = alloc(x);
49     if (k <= s[c[x][0]]) {

```

```

50     std::pair<int, int> ret = split(c[newnode][0], k);
51     c[newnode][0] = ret.second;
52     update(newnode);
53     return std::make_pair(ret.first, newnode);
54 }
55 else{
56     std::pair<int, int> ret = split(c[newnode][1], k - s[c[x][0]] - 1);
57     c[newnode][1] = ret.first;
58     update(newnode);
59     return std::make_pair(newnode, ret.second);
60 }
61 }
62
63 void travel(int x) {
64     if (c[x][0]) travel(c[x][0]);
65     putchar(d[x]);
66     if (d[x] == 'c') cnt++;
67     if (c[x][1]) travel(c[x][1]);
68 }
69
70 int build(int l, int r) {
71     int newnode = alloc();
72     d[newnode] = tmp[l + r >> 1];
73     if (l <= (l + r >> 1) - 1) c[newnode][0] = build(l, (l + r >> 1) - 1);
74     if ((l + r >> 1) + 1 <= r) c[newnode][1] = build((l + r >> 1) + 1, r);
75     update(newnode);
76     return newnode;
77 }
78
79 int main() {
80     scanf("%d", &n);
81     for (int i = 1, last = 0; i <= n; i++) {
82         int op, v, p, l;
83         scanf("%d", &op);
84         if (op == 1) {
85             scanf("%d%s", &p, tmp + 1);
86             p -= cnt;
87             std::pair<int, int> ret = split(rt[last], p);
88             rt[last + 1] = merge(std::make_pair(ret.first, build(1, strlen(tmp + 1))));
89             rt[last + 1] = merge(std::make_pair(rt[last + 1], ret.second));
90             last++;
91         }
92         else if (op == 2) {
93             scanf("%d%d", &p, &l);
94             p -= cnt; l -= cnt;
95             std::pair<int, int> A = split(rt[last], p - 1);
96             std::pair<int, int> B = split(A.second, l);
97             rt[last + 1] = merge(std::make_pair(A.first, B.second));
98             last++;
99         }
100         else if (op == 3) {
101             scanf("%d%d%d", &v, &p, &l);
102             v -= cnt; p -= cnt; l -= cnt;

```

```

103         std::pair<int, int> A = split(rt[v], p - 1);
104         std::pair<int, int> B = split(A.second, 1);
105         travel(B.first);
106         puts("");
107     }
108 }
109 return 0;
110 }

```

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。

```

1  struct Heapnode{
2      long long d;
3      int pos;
4      bool operator <(const Heapnode &p) const {
5          return d > p.d || (d == p.d && pos < p.pos);
6      }
7  };
8
9  struct MsgNode{
10     int xmin, xmax, ymin, ymax;
11     MsgNode() {}
12     MsgNode(const Point &a) : xmin(a.x), xmax(a.x), ymin(a.y), ymax(a.y) {}
13     long long dist(const Point &a) {
14         int dx = std::max(std::abs(a.x - xmin), std::abs(a.x - xmax));
15         int dy = std::max(std::abs(a.y - ymin), std::abs(a.y - ymax));
16         return (long long)dx * dx + (long long)dy * dy;
17     }
18     MsgNode operator +(const MsgNode &rhs) const {
19         MsgNode ret;
20         ret.xmin = std::min(xmin, rhs.xmin);
21         ret.xmax = std::max(xmax, rhs.xmax);
22         ret.ymin = std::min(ymin, rhs.ymin);
23         ret.ymax = std::max(ymax, rhs.ymax);
24         return ret;
25     }
26 };
27
28 struct TNode{
29     int l, r;
30     Point p;
31     MsgNode d;
32 }tree[MAXN];
33
34 void buildtree(int &rt, int l, int r, int pivot) {
35     if (l > r) return;
36     rt = ++size;
37     int mid = l + r >> 1;
38     if (pivot == 1) std::nth_element(p + l, p + mid, p + r + 1, cmpx);

```

```

39     if (pivot == 0) std::nth_element(p + 1, p + mid, p + r + 1, cmpy);
40     tree[rt].d = MsgNode(tree[rt].p = p[mid]);
41     buildtree(tree[rt].l, 1, mid - 1, pivot ^ 1);
42     buildtree(tree[rt].r, mid + 1, r, pivot ^ 1);
43     if (tree[rt].l) tree[rt].d = tree[rt].d + tree[tree[rt].l].d;
44     if (tree[rt].r) tree[rt].d = tree[rt].d + tree[tree[rt].r].d;
45 }
46
47 void query(int rt, const Point &a, int k, int pivot) {
48     Heapnode now = (Heapnode){dist(a, tree[rt].p), tree[rt].p.pos};
49     if (heap.size() < k) heap.push(now);
50     else if (now < heap.top()) {heap.pop(); heap.push(now);}
51     int lson = tree[rt].l, rson = tree[rt].r;
52     if (pivot == 1 && cmpx(a, tree[rt].p)) std::swap(lson, rson);
53     if (pivot == 0 && cmpy(a, tree[rt].p)) std::swap(lson, rson);
54     if (lson && (heap.size() < k || tree[lson].d.dist(a) >= heap.top().d)) query(lson, a, k,
55         pivot ^ 1);
56     if (rson && (heap.size() < k || tree[rson].d.dist(a) >= heap.top().d)) query(rson, a, k,
57         pivot ^ 1);
58 }
59
60 int main() {
61     for (int i = 1; i <= q; i++) {
62         int k;
63         Point now;
64         now.read();
65         scanf("%d", &k);
66         while (!heap.empty()) heap.pop();
67         query(rt, now, k, 1);
68         printf("%d\n", heap.top().pos);
69     }
70     return 0;
71 }

```

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。

```

1 struct Node{
2     int asum, lsum, rsum, zsum;
3     Node() {
4         asum = 0;
5         lsum = -INF;
6         rsum = -INF;
7         zsum = -INF;
8     }
9     Node(int d) : asum(d), lsum(d), rsum(d), zsum(d) {}
10    Node operator +(const Node &rhs) const {
11        Node ret;
12        ret.asum = asum + rhs.asum;

```

```

13         ret.lsum = std::max(lsum, asum + rhs.lsum);
14         ret.rsum = std::max(rsum + rhs.asum, rhs.rsum);
15         ret.zsum = std::max(zsum, rhs.zsum);
16         ret.zsum = std::max(ret.zsum, rsum + rhs.lsum);
17         return ret;
18     }
19 }tree[MAXN * 6];
20
21 int n, q, cnt, tot, h[MAXN], d[MAXN], t[MAXN], f[MAXN], s[MAXN], z[MAXN], w[MAXN], o[MAXN], a[
    MAXN];
22 std::pair<bool, int> flag[MAXN * 6];
23
24 void addedge(int x, int y) {
25     cnt++; e[cnt] = (Edge){y, h[x]}; h[x] = cnt;
26     cnt++; e[cnt] = (Edge){x, h[y]}; h[y] = cnt;
27 }
28
29 void makesame(int n, int l, int r, int d) {
30     flag[n] = std::make_pair(true, d);
31     tree[n].asum = d * (r - l + 1);
32     if (d > 0) {
33         tree[n].lsum = d * (r - l + 1);
34         tree[n].rsum = d * (r - l + 1);
35         tree[n].zsum = d * (r - l + 1);
36     }
37     else{
38         tree[n].lsum = d;
39         tree[n].rsum = d;
40         tree[n].zsum = d;
41     }
42 }
43
44 void pushdown(int n, int l, int r) {
45     if (flag[n].first) {
46         makesame(n << 1, l, l + r >> 1, flag[n].second);
47         makesame(n << 1 ^ 1, (l + r >> 1) + 1, r, flag[n].second);
48         flag[n] = std::make_pair(false, 0);
49     }
50 }
51
52 void modify(int n, int l, int r, int x, int y, int d) {
53     if (x <= l && r <= y) {
54         makesame(n, l, r, d);
55         return;
56     }
57     pushdown(n, l, r);
58     if ((l + r >> 1) < x) modify(n << 1 ^ 1, (l + r >> 1) + 1, r, x, y, d);
59     else if ((l + r >> 1) + 1 > y) modify(n << 1, l, l + r >> 1, x, y, d);
60     else{
61         modify(n << 1, l, l + r >> 1, x, y, d);
62         modify(n << 1 ^ 1, (l + r >> 1) + 1, r, x, y, d);
63     }
64     tree[n] = tree[n << 1] + tree[n << 1 ^ 1];

```

```

65 }
66
67 Node query(int n, int l, int r, int x, int y) {
68     if (x <= l && r <= y) return tree[n];
69     pushdown(n, l, r);
70     if ((l + r >> 1) < x) return query(n << 1 ^ 1, (l + r >> 1) + 1, r, x, y);
71     else if ((l + r >> 1) + 1 > y) return query(n << 1, l, l + r >> 1, x, y);
72     else{
73         Node left = query(n << 1, l, l + r >> 1, x, y);
74         Node right = query(n << 1 ^ 1, (l + r >> 1) + 1, r, x, y);
75         return left + right;
76     }
77 }
78
79 void modify(int x, int y, int val) {
80     int fx = t[x], fy = t[y];
81     while (fx != fy) {
82         if (d[fx] > d[fy]) {
83             modify(1, 1, n, w[fx], w[x], val);
84             x = f[fx]; fx = t[x];
85         }
86         else{
87             modify(1, 1, n, w[fy], w[y], val);
88             y = f[fy]; fy = t[y];
89         }
90     }
91     if (d[x] < d[y]) modify(1, 1, n, w[x], w[y], val);
92     else modify(1, 1, n, w[y], w[x], val);
93 }
94
95 Node query(int x, int y) {
96     int fx = t[x], fy = t[y];
97     Node left = Node(), right = Node();
98     while (fx != fy) {
99         if (d[fx] > d[fy]) {
100             left = query(1, 1, n, w[fx], w[x]) + left;
101             x = f[fx]; fx = t[x];
102         }
103         else{
104             right = query(1, 1, n, w[fy], w[y]) + right;
105             y = f[fy]; fy = t[y];
106         }
107     }
108     if (d[x] < d[y]) {
109         right = query(1, 1, n, w[x], w[y]) + right;
110     }
111     else{
112         left = query(1, 1, n, w[y], w[x]) + left;
113     }
114     std::swap(left.lsum, left.rsum);
115     return left + right;
116 }
117

```

```

118 void predfs(int x) {
119     s[x] = 1; z[x] = 0;
120     for (int i = h[x]; i; i = e[i].next) {
121         if (e[i].node == f[x]) continue;
122         f[e[i].node] = x;
123         d[e[i].node] = d[x] + 1;
124         predfs(e[i].node);
125         s[x] += s[e[i].node];
126         if (s[z[x]] < s[e[i].node]) z[x] = e[i].node;
127     }
128 }
129
130 void getanc(int x, int anc) {
131     t[x] = anc; w[x] = ++tot; o[tot] = x;
132     if (z[x]) getanc(z[x], anc);
133     for (int i = h[x]; i; i = e[i].next) {
134         if (e[i].node == f[x] || e[i].node == z[x]) continue;
135         getanc(e[i].node, e[i].node);
136     }
137 }
138
139 void buildtree(int n, int l, int r) {
140     if (l == r) {
141         tree[n] = Node(a[o[l]]);
142         return;
143     }
144     buildtree(n << 1, l, l + r >> 1);
145     buildtree(n << 1 ^ 1, (l + r >> 1) + 1, r);
146     tree[n] = tree[n << 1] + tree[n << 1 ^ 1];
147 }
148
149 int main() {
150     scanf("%d", &n);
151     for (int i = 1; i <= n; i++) scanf("%d", a + i);
152     for (int i = 1; i < n; i++) {
153         int x, y; scanf("%d%d", &x, &y);
154         addedge(x, y);
155     }
156     predfs(1);
157     getanc(1, 1);
158     buildtree(1, 1, n);
159     scanf("%d", &q);
160     for (int i = 1; i <= q; i++) {
161         int op, x, y, c;
162         scanf("%d", &op);
163         if (op == 1) {
164             scanf("%d%d", &x, &y);
165             Node ret = query(x, y);
166             printf("%d\n", std::max(0, ret.zsum));
167         }
168         else{
169             scanf("%d%d%d", &x, &y, &c);
170             modify(x, y, c);

```



```

171     }
172 }
173 return 0;
174 }

```

3.5.2 链操作版本

```

1 void modify(int x, int y) {
2     int fx = t[x], fy = t[y];
3     while (fx != fy) {
4         if (d[fx] > d[fy]) {
5             modify(1, 1, n, w[fx], w[x]);
6             x = f[fx]; fx = t[x];
7         }
8         else{
9             modify(1, 1, n, w[fy], w[y]);
10            y = f[fy]; fy = t[y];
11        }
12    }
13    if (x != y) {
14        if (d[x] < d[y]) modify(1, 1, n, w[z[x]], w[y]);
15        else modify(1, 1, n, w[z[y]], w[x]);
16    }
17 }

```

3.6 Link-Cut-Tree

```

1 struct MsgNode{
2     int leftColor, rightColor, answer;
3     MsgNode() {
4         leftColor = -1;
5         rightColor = -1;
6         answer = 0;
7     }
8     MsgNode(int c) {
9         leftColor = rightColor = c;
10        answer = 1;
11    }
12    MsgNode operator +(const MsgNode &p) const {
13        if (answer == 0) return p;
14        if (p.answer == 0) return *this;
15        MsgNode ret;
16        ret.leftColor = leftColor;
17        ret.rightColor = p.rightColor;
18        ret.answer = answer + p.answer - (rightColor == p.leftColor);
19        return ret;
20    }
21 }d[MAXN], g[MAXN];
22 int n, m, c[MAXN][2], f[MAXN], p[MAXN], s[MAXN], flag[MAXN];
23 bool r[MAXN];

```

```

24 void init(int x, int value) {
25     d[x] = g[x] = MsgNode(value);
26     c[x][0] = c[x][1] = 0;
27     f[x] = p[x] = flag[x] = -1;
28     s[x] = 1;
29 }
30 void update(int x) {
31     s[x] = s[c[x][0]] + s[c[x][1]] + 1;
32     g[x] = MsgNode();
33     if (c[x][0 ^ r[x]]) g[x] = g[x] + g[c[x][0 ^ r[x]]];
34     g[x] = g[x] + d[x];
35     if (c[x][1 ^ r[x]]) g[x] = g[x] + g[c[x][1 ^ r[x]]];
36 }
37 void makesame(int x, int c) {
38     flag[x] = c;
39     d[x] = MsgNode(c);
40     g[x] = MsgNode(c);
41 }
42 void pushdown(int x) {
43     if (r[x]) {
44         std::swap(c[x][0], c[x][1]);
45         r[c[x][0]] ^= 1;
46         r[c[x][1]] ^= 1;
47         std::swap(g[c[x][0]].leftColor, g[c[x][0]].rightColor);
48         std::swap(g[c[x][1]].leftColor, g[c[x][1]].rightColor);
49         r[x] = false;
50     }
51     if (flag[x] != -1) {
52         if (c[x][0]) makesame(c[x][0], flag[x]);
53         if (c[x][1]) makesame(c[x][1], flag[x]);
54         flag[x] = -1;
55     }
56 }
57 void rotate(int x, int k) {
58     pushdown(x); pushdown(c[x][k]);
59     int y = c[x][k]; c[x][k] = c[y][k ^ 1]; c[y][k ^ 1] = x;
60     if (f[x] != -1) c[f[x]][c[f[x]][1] == x] = y;
61     f[y] = f[x]; f[x] = y; f[c[x][k]] = x; std::swap(p[x], p[y]);
62     update(x); update(y);
63 }
64 void splay(int x, int s = -1) {
65     pushdown(x);
66     while (f[x] != s) {
67         if (f[f[x]] != s) rotate(f[f[x]], (c[f[f[x]]][1] == f[x]) ^ r[f[f[x]]]);
68         rotate(f[x], (c[f[x]][1] == x) ^ r[f[x]]);
69     }
70     update(x);
71 }
72 void access(int x) {
73     int y = 0;
74     while (x != -1) {
75         splay(x); pushdown(x);
76         f[c[x][1]] = -1; p[c[x][1]] = x;

```

```
77         c[x][1] = y; f[y] = x; p[y] = -1;
78         update(x); x = p[y = x];
79     }
80 }
81 void setroot(int x) {
82     access(x);
83     splay(x);
84     r[x] ^= 1;
85     std::swap(g[x].leftColor, g[x].rightColor);
86 }
87 void link(int x, int y) {
88     setroot(x);
89     p[x] = y;
90 }
```

Chapter 4

图论

4.1 强连通分量

```
1  int stamp, comps, top;
2  int dfn[N], low[N], comp[N], stack[N];
3
4  void tarjan(int x) {
5      dfn[x] = low[x] = ++stamp;
6      stack[top++] = x;
7      for (int i = 0; i < (int)edge[x].size(); ++i) {
8          int y = edge[x][i];
9          if (!dfn[y]) {
10             tarjan(y);
11             low[x] = std::min(low[x], low[y]);
12         } else if (!comp[y]) {
13             low[x] = std::min(low[x], dfn[y]);
14         }
15     }
16     if (low[x] == dfn[x]) {
17         comps++;
18         do {
19             int y = stack[--top];
20             comp[y] = comps;
21         } while (stack[top] != x);
22     }
23 }
24
25 void solve() {
26     stamp = comps = top = 0;
27     std::fill(dfn, dfn + n, 0);
28     std::fill(comp, comp + n, 0);
29     for (int i = 0; i < n; ++i) {
30         if (!dfn[i]) {
31             tarjan(i);
32         }
33     }
34 }
```

4.2 点双连通分量

4.2.1 坚固的点双连通分量

```

1  int n, m, x, y, ans1, ans2, tot1, tot2, flag, size, ind2, dfn[N], low[N], block[M], vis[N];
2  vector<int> a[N];
3  pair<int, int> stack[M];
4  void tarjan(int x, int p) {
5      dfn[x] = low[x] = ++ind2;
6      for (int i = 0; i < a[x].size(); ++i)
7          if (dfn[x] > dfn[a[x][i]] && a[x][i] != p){
8              stack[++size] = make_pair(x, a[x][i]);
9              if (i == a[x].size() - 1 || a[x][i] != a[x][i + 1])
10                 if (!dfn[a[x][i]]){
11                     tarjan(a[x][i], x);
12                     low[x] = min(low[x], low[a[x][i]]);
13                     if (low[a[x][i]] >= dfn[x]){
14                         tot1 = tot2 = 0;
15                         ++flag;
16                         for (; ; ){
17                             if (block[stack[size].first] != flag) {
18                                 ++tot1;
19                                 block[stack[size].first] = flag;
20                             }
21                             if (block[stack[size].second] != flag) {
22                                 ++tot1;
23                                 block[stack[size].second] = flag;
24                             }
25                             if (stack[size].first == x && stack[size].second == a[x][i])
26                                 break;
27                             ++tot2;
28                             --size;
29                         }
30                         for (; stack[size].first == x && stack[size].second == a[x][i]; --size)
31                             ++tot2;
32                         if (tot2 < tot1)
33                             ans1 += tot2;
34                         if (tot2 > tot1)
35                             ans2 += tot2;
36                     }
37                 }
38             else
39                 low[x] = min(low[x], dfn[a[x][i]]);
40         }
41 }
42 int main(){
43     for (; ; ){
44         scanf("%d%d", &n, &m);
45         if (n == 0 && m == 0) return 0;
46         for (int i = 1; i <= n; ++i) {
47             a[i].clear();
48             dfn[i] = 0;

```

```

49     }
50     for (int i = 1; i <= m; ++i){
51         scanf("%d%d",&x, &y);
52         ++x, ++y;
53         a[x].push_back(y);
54         a[y].push_back(x);
55     }
56     for (int i = 1; i <= n; ++i)
57         sort(a[i].begin(), a[i].end());
58     ans1 = ans2 = ind2 = 0;
59     for (int i = 1; i <= n; ++i)
60         if (!dfn[i]) {
61             size = 0;
62             tarjan(i, 0);
63         }
64     printf("%d_ %d\n", ans1, ans2);
65 }
66 return 0;
67 }

```

4.2.2 朴素的点双连通分量

```

1 void tarjan(int x){
2     dfn[x] = low[x] = ++ind2;
3     v[x] = 1;
4     for (int i = nt[x]; pt[i]; i = nt[i])
5         if (!dfn[pt[i]]){
6             tarjan(pt[i]);
7             low[x] = min(low[x], low[pt[i]]);
8             if (dfn[x] <= low[pt[i]])
9                 ++v[x];
10        }
11        else
12            low[x] = min(low[x], dfn[pt[i]]);
13 }
14 int main(){
15     for (; ; ){
16         scanf("%d%d", &n, &m);
17         if (n == 0 && m == 0)
18             return 0;
19         for (int i = 1; i <= ind; ++i)
20             nt[i] = pt[i] = 0;
21         ind = n;
22         for (int i = 1; i <= ind; ++i)
23             last[i] = i;
24         for (int i = 1; i <= m; ++i){
25             scanf("%d%d", &x, &y);
26             ++x, ++y;
27             edge(x, y), edge(y, x);
28         }
29         memset(dfn, 0, sizeof(dfn));
30         memset(v, 0, sizeof(v));

```

```

31     ans = num = ind2 = 0;
32     for (int i = 1; i <= n; ++i)
33         if (!dfn[i]){
34             root = i;
35             size = 0;
36             ++num;
37             tarjan(i);
38             —v[root];
39         }
40     for (int i = 1; i <= n; ++i)
41         if (v[i] + num - 1 > ans)
42             ans = v[i] + num - 1;
43     printf("%d\n", ans);
44 }
45 return 0;
46 }

```

4.3 2-SAT 问题

```

1  int stamp, comps, top;
2  int dfn[N], low[N], comp[N], stack[N];
3
4  void add(int x, int a, int y, int b) {
5      edge[x << 1 | a].push_back(y << 1 | b);
6  }
7
8  void tarjan(int x) {
9      dfn[x] = low[x] = ++stamp;
10     stack[top++] = x;
11     for (int i = 0; i < (int)edge[x].size(); ++i) {
12         int y = edge[x][i];
13         if (!dfn[y]) {
14             tarjan(y);
15             low[x] = std::min(low[x], low[y]);
16         } else if (!comp[y]) {
17             low[x] = std::min(low[x], dfn[y]);
18         }
19     }
20     if (low[x] == dfn[x]) {
21         comps++;
22         do {
23             int y = stack[--top];
24             comp[y] = comps;
25         } while (stack[top] != x);
26     }
27 }
28
29 bool solve() {
30     int counter = n + n + 1;
31     stamp = top = comps = 0;
32     std::fill(dfn, dfn + counter, 0);

```

```

33     std::fill(comp, comp + counter, 0);
34     for (int i = 0; i < counter; ++i) {
35         if (!dfn[i]) {
36             tarjan(i);
37         }
38     }
39     for (int i = 0; i < n; ++i) {
40         if (comp[i << 1] == comp[i << 1 | 1]) {
41             return false;
42         }
43         answer[i] = (comp[i << 1 | 1] < comp[i << 1]);
44     }
45     return true;
46 }

```

4.4 二分图最大匹配

4.4.1 Hungary 算法

时间复杂度: $\mathcal{O}(V \cdot E)$

```

1  int n, m, stamp;
2  int match[N], visit[N];
3
4  bool dfs(int x) {
5      for (int i = 0; i < (int)edge[x].size(); ++i) {
6          int y = edge[x][i];
7          if (visit[y] != stamp) {
8              visit[y] = stamp;
9              if (match[y] == -1 || dfs(match[y])) {
10                 match[y] = x;
11                 return true;
12             }
13         }
14     }
15     return false;
16 }
17
18 int solve() {
19     std::fill(match, match + m, -1);
20     int answer = 0;
21     for (int i = 0; i < n; ++i) {
22         stamp++;
23         answer += dfs(i);
24     }
25     return answer;
26 }

```

4.4.2 Hopcroft Karp 算法

时间复杂度: $\mathcal{O}(\sqrt{V} \cdot E)$


```

1  int matchx[N], matchy[N], level[N];
2
3  bool dfs(int x) {
4      for (int i = 0; i < (int)edge[x].size(); ++i) {
5          int y = edge[x][i];
6          int w = matchy[y];
7          if (w == -1 || level[x] + 1 == level[w] && dfs(w)) {
8              matchx[x] = y;
9              matchy[y] = x;
10             return true;
11         }
12     }
13     level[x] = -1;
14     return false;
15 }
16
17 int solve() {
18     std::fill(matchx, matchx + n, -1);
19     std::fill(matchy, matchy + m, -1);
20     for (int answer = 0; ; ) {
21         std::vector<int> queue;
22         for (int i = 0; i < n; ++i) {
23             if (matchx[i] == -1) {
24                 level[i] = 0;
25                 queue.push_back(i);
26             } else {
27                 level[i] = -1;
28             }
29         }
30         for (int head = 0; head < (int)queue.size(); ++head) {
31             int x = queue[head];
32             for (int i = 0; i < (int)edge[x].size(); ++i) {
33                 int y = edge[x][i];
34                 int w = matchy[y];
35                 if (w != -1 && level[w] < 0) {
36                     level[w] = level[x] + 1;
37                     queue.push_back(w);
38                 }
39             }
40         }
41         int delta = 0;
42         for (int i = 0; i < n; ++i) {
43             if (matchx[i] == -1 && dfs(i)) {
44                 delta++;
45             }
46         }
47         if (delta == 0) {
48             return answer;
49         } else {
50             answer += delta;
51         }
52     }
53 }

```

4.5 二分图最大权匹配

时间复杂度: $\mathcal{O}(V^4)$

```

1  int DFS(int x){
2      visx[x] = 1;
3      for (int y = 1; y <= ny; y++){
4          if (visy[y]) continue;
5          int t = lx[x] + ly[y] - w[x][y];
6          if (t == 0) {
7              visy[y] = 1;
8              if (link[y] == -1 || DFS(link[y])){
9                  link[y] = x;
10                 return 1;
11             }
12         }
13         else slack[y] = min(slack[y], t);
14     }
15     return 0;
16 }
17 int KM(){
18     int i, j;
19     memset(link, -1, sizeof(link));
20     memset(ly, 0, sizeof(ly));
21     for (i = 1; i <= nx; i++){
22         for (j = 1, lx[i] = -inf; j <= ny; j++){
23             lx[i] = max(lx[i], w[i][j]);
24         }
25         for (int x = 1; x <= nx; x++){
26             for (i = 1; i <= ny; i++) slack[i] = inf;
27             while (true) {
28                 memset(visx, 0, sizeof(visx));
29                 memset(visy, 0, sizeof(visy));
30                 if (DFS(x)) break;
31                 int d = inf;
32                 for (i = 1; i <= ny; i++)
33                     if (!visy[i] && d > slack[i]) d = slack[i];
34                 for (i = 1; i <= nx; i++)
35                     if (visx[i]) lx[i] -= d;
36                 for (i = 1; i <= ny; i++)
37                     if (visy[i]) ly[i] += d;
38                     else slack[i] -= d;
39             }
40         }
41         int res = 0;
42         for (i = 1; i <= ny; i++)
43             if (link[i] > -1) res += w[link[i]][i];
44         return res;
45     }
46 }
```

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: $\mathcal{O}(V^2 \cdot E)$

```

1  int bfs(){
2      for (int i = 1; i <= t; i++) d[i] = -1;
3      int l, r;
4      q[l = r = 0] = s, d[s] = 0;
5      for (; l <= r; l++)
6          for (int k = h[q[l]]; k > -1; k = nxt[k])
7              if (d[p[k]] == -1 && c[k] > 0) d[p[k]] = d[q[l]] + 1, q[++ r] = p[k];
8      return d[t] > -1 ? 1 : 0;
9  }
10 int dfs(int u, int ext){
11     if (u == t) return ext;
12     int k = w[u], ret = 0;
13     for (; k > -1; k = nxt[k], w[u] = k){          //w数组为当前弧
14         if (ext == 0) break;
15         if (d[p[k]] == d[u] + 1 && c[k] > 0){
16             int flow = dfs(p[k], min(c[k], ext));
17             if (flow > 0){
18                 c[k] -= flow, c[k ^ 1] += flow;
19                 ret += flow, ext -= flow;          //ret累计增广量, ext记录还可增广的量
20             }
21         }
22     }
23     if (k == -1) d[u] = -1;
24     return ret;
25 }
26 void dinic() {
27     while (bfs()) {
28         for (int i = 1; i <= t; i++) w[i] = h[i];
29         dfs(s, inf);
30     }
31 }

```

4.6.2 ISAP

时间复杂度: $\mathcal{O}(V^2 \cdot E)$

```

1  int Maxflow_Isap(int s, int t, int n) {
2      std::fill(pre + 1, pre + n + 1, 0);
3      std::fill(d + 1, d + n + 1, 0);
4      std::fill(gap + 1, gap + n + 1, 0);
5      for (int i = 1; i <= n; i++) cur[i] = h[i];
6      gap[0] = n;
7      int u = pre[s] = s, v, maxflow = 0;
8      while (d[s] < n) {
9          v = n + 1;

```

```

10     for (int i = cur[u]; i; i = e[i].next)
11     if (e[i].flow && d[u] == d[e[i].node] + 1) {
12         v = e[i].node; cur[u]=i; break;
13     }
14     if (v <= n) {
15         pre[v] = u; u = v;
16         if (v == t) {
17             int dflow = INF, p = t; u = s;
18             while (p != s) {
19                 p = pre[p];
20                 dflow = std::min(dflow, e[cur[p]].flow);
21             }
22             maxflow += dflow; p = t;
23             while (p != s) {
24                 p = pre[p];
25                 e[cur[p]].flow -= dflow;
26                 e[e[cur[p]].opp].flow += dflow;
27             }
28         }
29     }
30     else{
31         int mindist = n + 1;
32         for (int i = h[u]; i; i = e[i].next)
33             if (e[i].flow && mindist > d[e[i].node]) {
34                 mindist = d[e[i].node]; cur[u] = i;
35             }
36         if (!--gap[d[u]]) return maxflow;
37         gap[d[u] = mindist + 1]++; u = pre[u];
38     }
39 }
40 return maxflow;
41 }

```

4.6.3 SAP

时间复杂度: $\mathcal{O}(V^2 \cdot E)$

```

1  const int N = 110, M = 30110, INF = 1000000000; //边表不要开小
2  int n, m, ind, S, T, flow, tot, pt[M], nt[M], last[N], size[M], num[N], h[N], now[N];
3  void edge(int x, int y, int z){
4      last[x] = nt[last[x]] = ++ind;
5      pt[ind] = y, size[ind] = z;
6  }
7  int aug(int x, int y){
8      if (x == T)
9          return y;
10     int f = y;
11     for (int i = now[x]; pt[i]; i = nt[i])
12         if (size[i] && h[pt[i]] + 1 == h[x]){
13             int z = aug(pt[i], min(f, size[i]));
14             f -= z;
15             size[i] -= z;
16             size[i ^ 1] += z;

```

```

17         now[x] = i;
18         if (h[S] > tot || f == 0)
19             return y - f;
20     }
21     now[x] = nt[x];
22     if (--num[h[x]] == 0)
23         h[S] = tot + 1;
24     ++num[++h[x]];
25     return y - f;
26 }
27 int main(){
28     int np, nc;
29     for (; scanf("%d%d%d", &n, &np, &nc, &m) == 4; ) {
30         for (int i = 0; i <= ind; ++i)
31             pt[i] = nt[i] = last[i] = size[i] = 0;
32         ind = n + 2;
33         if (ind % 2 == 0)
34             ++ind;
35         S = n + 1, tot = T = n + 2;
36         for (int i = 0; i <= tot; ++i)
37             num[i] = h[i] = now[i] = 0;
38         for (int i = 1; i <= tot; ++i)
39             last[i] = i;
40         for (int i = 1; i <= m; ++i){
41             int x, y, z;
42             for (; getchar() != '('; );
43             scanf("%d%c%d%c%d", &x, &y, &z);
44             ++x, ++y;
45             edge(x, y, z);
46             edge(y, x, 0);
47         }
48         for (int i = 1; i <= np; ++i) {
49             int y, z;
50             for (; getchar() != '('; );
51             scanf("%d%c%d", &y, &z);
52             ++y;
53             edge(S, y, z);
54             edge(y, S, 0);
55         }
56         for (int i = 1; i <= nc; ++i) {
57             int x, z;
58             for (; getchar() != '('; );
59             scanf("%d%c%d", &x, &z);
60             ++x;
61             edge(x, T, z);
62             edge(T, x, 0);
63         }
64         num[0] = tot;
65         for (int i = 1; i <= tot; ++i)
66             now[i] = nt[i];
67         flow = 0;
68         for (; h[S] <= T; )
69             flow += aug(S, INF);

```

```

70     printf("%d\n", flow);
71 }
72 return 0;
73 }
```

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

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

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: $S^* \rightarrow v$, 容量为 $B(u, v)$; $u \rightarrow T^*$, 容量为 $B(u, v)$; $u \rightarrow v$, 容量为 $C(u, v) - B(u, v)$ 。最后求新网络的最大流, 判断从超级源点 S^* 出发的边是否都满流即可, 边 (u, v) 的最终解中的实际流量为 $G(u, v) + B(u, v)$ 。

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

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

4.8 最小费用最大流

4.8.1 稀疏图

时间复杂度: $\mathcal{O}(V \cdot E^2)$

```

1  struct EdgeList {
2      int size;
3      int last[N];
4      int succ[M], other[M], flow[M], cost[M];
5      void clear(int n) {
6          size = 0;
7          std::fill(last, last + n, -1);
8      }
9      void add(int x, int y, int c, int w) {
10         succ[size] = last[x];
11         last[x] = size;
12         other[size] = y;
13         flow[size] = c;
14         cost[size++] = w;
15     }
16 } e;
17
18 int n, source, target;
19 int prev[N];
20
21 void add(int x, int y, int c, int w) {
22     e.add(x, y, c, w);
23     e.add(y, x, 0, -w);
24 }
25
26 bool augment() {
27     static int dist[N], occur[N];
28     std::vector<int> queue;
29     std::fill(dist, dist + n, INT_MAX);
30     std::fill(occur, occur + n, 0);
31     dist[source] = 0;
32     occur[source] = true;
33     queue.push_back(source);
34     for (int head = 0; head < (int)queue.size(); ++head) {
35         int x = queue[head];
36         for (int i = e.last[x]; ~i; i = e.succ[i]) {
37             int y = e.other[i];
38             if (e.flow[i] && dist[y] > dist[x] + e.cost[i]) {
39                 dist[y] = dist[x] + e.cost[i];
40                 prev[y] = i;
41                 if (!occur[y]) {
42                     occur[y] = true;
43                     queue.push_back(y);
44                 }
45             }
46         }
47         occur[x] = false;
48     }
49     return dist[target] < INT_MAX;
50 }
51
52 std::pair<int, int> solve() {
53     std::pair<int, int> answer = std::make_pair(0, 0);

```

```

54     while (augment()) {
55         int number = INT_MAX;
56         for (int i = target; i != source; i = e.other[prev[i] ^ 1]) {
57             number = std::min(number, e.flow[prev[i]]);
58         }
59         answer.first += number;
60         for (int i = target; i != source; i = e.other[prev[i] ^ 1]) {
61             e.flow[prev[i]] -= number;
62             e.flow[prev[i] ^ 1] += number;
63             answer.second += number * e.cost[prev[i]];
64         }
65     }
66     return answer;
67 }

```

4.8.2 稠密图

使用条件：费用非负

时间复杂度： $\mathcal{O}(V \cdot E^2)$

```

1  int aug(int no,int res) {
2      if(no == t) return cost += pil * res,res;
3      v[no] = true;
4      int flow = 0;
5      for(int i = h[no]; ~ i ;i = nxt[i])
6          if(cap[i] && !expense[i] && !v[p[i]]) {
7              int d = aug(p[i],min(res,cap[i]));
8              cap[i] -= d,cap[i ^ 1] += d,flow += d,res -= d;
9              if( !res ) return flow;
10         }
11     return flow;
12 }
13 bool modlabel() {
14     int d = maxint;
15     for(int i = 1;i <= t;++ i)
16         if(v[i]) {
17             for(int j = h[i]; ~ j ;j = nxt[j])
18                 if(cap[j] && !v[p[j]] && expense[j] < d) d = expense[j];
19         }
20     if(d == maxint) return false;
21     for(int i = 1;i <= t;++ i)
22         if(v[i]) {
23             for(int j = h[i]; ~ j ;j = nxt[j])
24                 expense[j] -= d,expense[j ^ 1] += d;
25         }
26     pil += d;
27     return true;
28 }
29 void minimum_cost_flow_zkw() {
30     cost = 0;
31     do{
32         do{
33             memset(v,false,sizeof v);

```



```

34         }while (aug(s,maxint));
35     }while (modlabel());
36 }

```

4.9 一般图最大匹配

时间复杂度: $\mathcal{O}(V^3)$

```

1  int match[N], belong[N], next[N], mark[N], visit[N];
2  std::vector<int> queue;
3
4  int find(int x) {
5      if (belong[x] != x) {
6          belong[x] = find(belong[x]);
7      }
8      return belong[x];
9  }
10
11 void merge(int x, int y) {
12     x = find(x);
13     y = find(y);
14     if (x != y) {
15         belong[x] = y;
16     }
17 }
18
19 int lca(int x, int y) {
20     static int stamp = 0;
21     stamp++;
22     while (true) {
23         if (x != -1) {
24             x = find(x);
25             if (visit[x] == stamp) {
26                 return x;
27             }
28             visit[x] = stamp;
29             if (match[x] != -1) {
30                 x = next[match[x]];
31             } else {
32                 x = -1;
33             }
34         }
35         std::swap(x, y);
36     }
37 }
38
39 void group(int a, int p) {
40     while (a != p) {
41         int b = match[a], c = next[b];
42         if (find(c) != p) {
43             next[c] = b;
44         }

```

```

45     if (mark[b] == 2) {
46         mark[b] = 1;
47         queue.push_back(b);
48     }
49     if (mark[c] == 2) {
50         mark[c] = 1;
51         queue.push_back(c);
52     }
53     merge(a, b);
54     merge(b, c);
55     a = c;
56 }
57 }
58
59 void augment(int source) {
60     queue.clear();
61     for (int i = 0; i < n; ++i) {
62         next[i] = visit[i] = -1;
63         belong[i] = i;
64         mark[i] = 0;
65     }
66     mark[source] = 1;
67     queue.push_back(source);
68     for (int head = 0; head < (int)queue.size() && match[source] == -1; ++head) {
69         int x = queue[head];
70         for (int i = 0; i < (int)edge[x].size(); ++i) {
71             int y = edge[x][i];
72             if (match[x] == y || find(x) == find(y) || mark[y] == 2) {
73                 continue;
74             }
75             if (mark[y] == 1) {
76                 int r = lca(x, y);
77                 if (find(x) != r) {
78                     next[x] = y;
79                 }
80                 if (find(y) != r) {
81                     next[y] = x;
82                 }
83                 group(x, r);
84                 group(y, r);
85             } else if (match[y] == -1) {
86                 next[y] = x;
87                 for (int u = y; u != -1; ) {
88                     int v = next[u];
89                     int mv = match[v];
90                     match[v] = u;
91                     match[u] = v;
92                     u = mv;
93                 }
94                 break;
95             } else {
96                 next[y] = x;
97                 mark[y] = 2;

```

4.10. 无向图全局最小割

```

98         mark[match[y]] = 1;
99         queue.push_back(match[y]);
100     }
101 }
102 }
103 }
104
105 int solve() {
106     std::fill(match, match + n, -1);
107     for (int i = 0; i < n; ++i) {
108         if (match[i] == -1) {
109             augment(i);
110         }
111     }
112     int answer = 0;
113     for (int i = 0; i < n; ++i) {
114         answer += (match[i] != -1);
115     }
116     return answer;
117 }

```

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

```

1  int node[N], dist[N];
2  bool visit[N];
3
4  int solve(int n) {
5      int answer = INT_MAX;
6      for (int i = 0; i < n; ++i) {
7          node[i] = i;
8      }
9      while (n > 1) {
10         int max = 1;
11         for (int i = 0; i < n; ++i) {
12             dist[node[i]] = graph[node[0]][node[i]];
13             if (dist[node[i]] > dist[node[max]]) {
14                 max = i;
15             }
16         }
17         int prev = 0;
18         memset(visit, 0, sizeof(visit));
19         visit[node[0]] = true;
20         for (int i = 1; i < n; ++i) {
21             if (i == n - 1) {
22                 answer = std::min(answer, dist[node[max]]);
23                 for (int k = 0; k < n; ++k) {
24                     graph[node[k]][node[prev]] =
25                         (graph[node[prev]][node[k]] += graph[node[k]][node[max]]);
26                 }

```

```

27         node[max] = node[--n];
28     }
29     visit[node[max]] = true;
30     prev = max;
31     max = -1;
32     for (int j = 1; j < n; ++j) {
33         if (!visit[node[j]]) {
34             dist[node[j]] += graph[node[prev]][node[j]];
35             if (max == -1 || dist[node[max]] < dist[node[j]]) {
36                 max = j;
37             }
38         }
39     }
40 }
41 }
42 return answer;
43 }

```

4.11 最小树形图

```

1  int n, m, used[N], pass[N], eg[N], more, queue[N];
2  double g[N][N];
3
4  void combine(int id, double &sum) {
5      int tot = 0, from, i, j, k;
6      for (; id != 0 && !pass[id]; id = eg[id]) {
7          queue[tot++] = id;
8          pass[id] = 1;
9      }
10
11     for (from = 0; from < tot && queue[from] != id; from++);
12     if (from == tot) return;
13     more = 1;
14     for (i = from; i < tot; i++) {
15         sum += g[eg[queue[i]]][queue[i]];
16         if (i != from) {
17             used[queue[i]] = 1;
18             for (j = 1; j <= n; j++) if (!used[j]) {
19                 if (g[queue[i]][j] < g[id][j]) g[id][j] = g[queue[i]][j];
20             }
21         }
22     }
23
24     for (i = 1; i <= n; i++) if (!used[i] && i != id) {
25         for (j = from; j < tot; j++) {
26             k = queue[j];
27             if (g[i][id] > g[i][k] - g[eg[k]][k]) g[i][id] = g[i][k] - g[eg[k]][k];
28         }
29     }
30 }
31

```

4.12. 有根树的同构

```

32 double mdst(int root) {
33     int i, j, k;
34     double sum = 0;
35     memset(used, 0, sizeof(used));
36     for (more = 1; more; ) {
37         more = 0;
38         memset(eg, 0, sizeof(eg));
39         for (i = 1; i <= n; i++) if (!used[i] && i != root) {
40             for (j = 1, k = 0; j <= n; j++) if (!used[j] && i != j)
41                 if (k == 0 || g[j][i] < g[k][i]) k = j;
42             eg[i] = k;
43         }
44
45         memset(pass, 0, sizeof(pass));
46         for (i = 1; i <= n; i++) if (!used[i] && !pass[i] && i != root) combine(i, sum);
47     }
48
49     for (i = 1; i <= n; i++) if (!used[i] && i != root) sum += g[eg[i]][i];
50     return sum;
51 }

```

4.12 有根树的同构

时间复杂度: $\mathcal{O}(V \log V)$

```

1  const unsigned long long MAGIC = 4423;
2
3  unsigned long long magic[N];
4  std::pair<unsigned long long, int> hash[N];
5
6  void solve(int root) {
7      magic[0] = 1;
8      for (int i = 1; i <= n; ++i) {
9          magic[i] = magic[i - 1] * MAGIC;
10     }
11     std::vector<int> queue;
12     queue.push_back(root);
13     for (int head = 0; head < (int)queue.size(); ++head) {
14         int x = queue[head];
15         for (int i = 0; i < (int)son[x].size(); ++i) {
16             int y = son[x][i];
17             queue.push_back(y);
18         }
19     }
20     for (int index = n - 1; index >= 0; --index) {
21         int x = queue[index];
22         hash[x] = std::make_pair(0, 0);
23
24         std::vector<std::pair<unsigned long long, int> > value;
25         for (int i = 0; i < (int)son[x].size(); ++i) {
26             int y = son[x][i];
27             value.push_back(hash[y]);

```

```

28     }
29     std::sort(value.begin(), value.end());
30
31     hash[x].first = hash[x].first * magic[1] + 37;
32     hash[x].second++;
33     for (int i = 0; i < (int)value.size(); ++i) {
34         hash[x].first = hash[x].first * magic[value[i].second] + value[i].first;
35         hash[x].second += value[i].second;
36     }
37     hash[x].first = hash[x].first * magic[1] + 41;
38     hash[x].second++;
39 }
40 }

```

4.13 度限制生成树

```

1  int n, m, S, K, ans, cnt, Best[N], fa[N], FE[N];
2  int f[N], p[M], t[M], c[M], o, Cost[N];
3  bool u[M], d[M];
4  pair<int, int> MinCost[N];
5  struct Edge {
6      int a, b, c;
7      bool operator < (const Edge & E) const { return c < E.c; }
8  }E[M];
9  vector<int> SE;
10 inline int F(int x) {
11     return fa[x] == x ? x : fa[x] = F(fa[x]);
12 }
13 inline void AddEdge(int a, int b, int C) {
14     p[++o] = b; c[o] = C;
15     t[o] = f[a]; f[a] = o;
16 }
17 void dfs(int i, int father) {
18     fa[i] = father;
19     if (father == S) Best[i] = -1;
20     else {
21         Best[i] = i;
22         if (~Best[father] && Cost[Best[father]] > Cost[i]) Best[i] = Best[father];
23     }
24     for (int j = f[i]; j; j = t[j])
25     if (!d[j] && p[j] != father) {
26         Cost[p[j]] = c[j];
27         FE[p[j]] = j;
28         dfs(p[j], i);
29     }
30 }
31 inline bool Kruskal() {
32     cnt = n - 1, ans = 0; o = 1;
33     for (int i = 1; i <= n; i++) fa[i] = i, f[i] = 0;
34     sort(E + 1, E + m + 1);
35     for (int i = 1; i <= m; i++) {

```

```

36     if (E[i].b == S) swap(E[i].a, E[i].b);
37     if (E[i].a != S && F(E[i].a) != F(E[i].b)) {
38         fa[F(E[i].a)] = F(E[i].b);
39         ans += E[i].c;
40         cnt--;
41         u[i] = true;
42         AddEdge(E[i].a, E[i].b, E[i].c);
43         AddEdge(E[i].b, E[i].a, E[i].c);
44     }
45 }
46 for (int i = 1; i <= n; i++) MinCost[i] = make_pair(INF, INF);
47 for (int i = 1; i <= m; i++)
48     if (E[i].a == S) {
49         SE.push_back(i);
50         MinCost[F(E[i].b)] = min(MinCost[F(E[i].b)], make_pair(E[i].c, i));
51     }
52 int dif = 0;
53 for (int i = 1; i <= n; i++)
54     if (i != S && fa[i] == i) {
55         if (MinCost[i].second == INF) return false;
56         if (++dif > K) return false;
57         dfs(E[MinCost[i].second].b, S);
58         u[MinCost[i].second] = true;
59         ans += MinCost[i].first;
60     }
61 return true;
62 }
63 bool Solve() {
64     memset(d, false, sizeof d);
65     memset(u, false, sizeof u);
66     if (!Kruskal()) return false;
67     for (int i = cnt + 1; i <= K && i <= n; i++) {
68         int MinD = INF, MinID = -1;
69         for (int j = (int) SE.size() - 1; j >= 0; j--)
70             if (u[SE[j]])
71                 SE.erase(SE.begin() + j);
72         for (int j = 0; j < (int) SE.size(); j++) {
73             int tmp = E[SE[j]].c - Cost[Best[E[SE[j]].b]];
74             if (tmp < MinD) {
75                 MinD = tmp;
76                 MinID = SE[j];
77             }
78         }
79         if (MinID == -1) return true;
80         if (MinD >= 0) break;
81         ans += MinD;
82         u[MinID] = true;
83         d[FE[Best[E[MinID].b]]] = d[FE[Best[E[MinID].b]] ^ 1] = true;
84         dfs(E[MinID].b, S);
85     }
86     return true;
87 }
88 int main() {

```

```

89     Solve();
90     return 0;
91 }

```

4.14 弦图相关

4.14.1 弦图的判定

```

1  int n, m, first[1001], l, next[2000001], where[2000001], f[1001], a[1001], c[1001], L[1001], R
    [1001],
2  v[1001], idx[1001], pos[1001];
3  bool b[1001][1001];
4
5  inline void makelist(int x, int y){
6      where[++l] = y;
7      next[l] = first[x];
8      first[x] = l;
9  }
10
11 bool cmp(const int &x, const int &y){
12     return(idx[x] < idx[y]);
13 }
14
15 int main(){
16     for (;;)
17     {
18         n = read(); m = read();
19         if (!n && !m) return 0;
20         memset(first, 0, sizeof(first)); l = 0;
21         memset(b, false, sizeof(b));
22         for (int i = 1; i <= m; i++)
23         {
24             int x = read(), y = read();
25             if (x != y && !b[x][y])
26             {
27                 b[x][y] = true; b[y][x] = true;
28                 makelist(x, y); makelist(y, x);
29             }
30         }
31         memset(f, 0, sizeof(f));
32         memset(L, 0, sizeof(L));
33         memset(R, 255, sizeof(R));
34         L[0] = 1; R[0] = n;
35         for (int i = 1; i <= n; i++) c[i] = i, pos[i] = i;
36         memset(idx, 0, sizeof(idx));
37         memset(v, 0, sizeof(v));
38         for (int i = n; i; --i)
39         {
40             int now = c[i];
41             R[f[now]]--;
42             if (R[f[now]] < L[f[now]]) R[f[now]] = -1;

```



```

43     idx[now] = i; v[i] = now;
44     for (int x = first[now]; x; x = next[x])
45         if (!idx[where[x]])
46             {
47                 swap(c[pos[where[x]]], c[R[f[where[x]]]]);
48                 pos[c[pos[where[x]]]] = pos[where[x]];
49                 pos[where[x]] = R[f[where[x]]];
50                 L[f[where[x]] + 1] = R[f[where[x]]]--;
51                 if (R[f[where[x]]] < L[f[where[x]]]) R[f[where[x]]] = -1;
52                 if (R[f[where[x]] + 1] == -1)
53                     R[f[where[x]] + 1] = L[f[where[x]] + 1];
54                 ++f[where[x]];
55             }
56     }
57     bool ok = true;
58     //v是完美消除序列.
59     for (int i = 1; i <= n && ok; i++)
60     {
61         int cnt = 0;
62         for (int x = first[v[i]]; x; x = next[x])
63             if (idx[where[x]] > i) c[++cnt] = where[x];
64         sort(c + 1, c + cnt + 1, cmp);
65         bool can = true;
66         for (int j = 2; j <= cnt; j++)
67             if (!b[c[1]][c[j]])
68                 {
69                     ok = false;
70                     break;
71                 }
72     }
73     if (ok) printf("Perfect\n");
74     else printf("Imperfect\n");
75     printf("\n");
76 }
77 }

```

4.14.2 弦图的团数

```

1  int n, m, first[100001], next[2000001], where[2000001], l, L[100001], R[100001], c[100001], f
    [100001],
2  pos[100001], idx[100001], v[100001], ans;
3
4  inline void makelist(int x, int y){
5      where[++l] = y;
6      next[l] = first[x];
7      first[x] = l;
8  }
9
10 int read(){
11     char ch;
12     for (ch = getchar(); ch < '0' || ch > '9'; ch = getchar());
13     int cnt = 0;

```

```

14     for (; ch >= '0' && ch <= '9'; ch = getchar()) cnt = cnt * 10 + ch - '0';
15     return(cnt);
16 }
17
18 int main(){
19     //freopen("1006.in", "r", stdin);
20     //freopen("1006.out", "w", stdout);
21     memset(first, 0, sizeof(first)); l = 0;
22     n = read(); m = read();
23     for (int i = 1; i <= m; i++)
24     {
25         int x, y;
26         x = read(); y = read();
27         makelist(x, y); makelist(y, x);
28     }
29     memset(L, 0, sizeof(L));
30     memset(R, 255, sizeof(R));
31     memset(f, 0, sizeof(f));
32     memset(idx, 0, sizeof(idx));
33     for (int i = 1; i <= n; i++) c[i] = i, pos[i] = i;
34     L[0] = 1; R[0] = n; ans = 0;
35     for (int i = n; i; --i)
36     {
37         int now = c[i], cnt = 1;
38         idx[now] = i; v[i] = now;
39         if (--R[f[now]] < L[f[now]]) R[f[now]] = -1;
40         for (int x = first[now]; x; x = next[x])
41             if (!idx[where[x]])
42             {
43                 swap(c[pos[where[x]]], c[R[f[where[x]]]]);
44                 pos[c[pos[where[x]]]] = pos[where[x]];
45                 pos[where[x]] = R[f[where[x]]];
46                 L[f[where[x]] + 1] = R[f[where[x]]]--;
47                 if (R[f[where[x]]] < L[f[where[x]]]) R[f[where[x]]] = -1;
48                 if (R[f[where[x]] + 1] == -1) R[f[where[x]] + 1] = L[f[where[x]] + 1];
49                 ++f[where[x]];
50             }
51         else ++cnt;
52         ans = max(ans, cnt);
53     }
54     printf("%d\n", ans);
55 }

```

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: $n \geq 3$

```

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

```

```

2
3 void cover(int x) {
4     left[right[x]] = left[x];
5     right[left[x]] = right[x];
6 }
7
8 int adjacent(int x) {
9     for (int i = right[0]; i <= n; i = right[i]) {
10         if (graph[x][i]) {
11             return i;
12         }
13     }
14     return 0;
15 }
16
17 std::vector<int> solve() {
18     for (int i = 1; i <= n; ++i) {
19         left[i] = i - 1;
20         right[i] = i + 1;
21     }
22     int head, tail;
23     for (int i = 2; i <= n; ++i) {
24         if (graph[1][i]) {
25             head = 1;
26             tail = i;
27             cover(head);
28             cover(tail);
29             next[head] = tail;
30             break;
31         }
32     }
33     while (true) {
34         int x;
35         while (x = adjacent(head)) {
36             next[x] = head;
37             head = x;
38             cover(head);
39         }
40         while (x = adjacent(tail)) {
41             next[tail] = x;
42             tail = x;
43             cover(tail);
44         }
45         if (!graph[head][tail]) {
46             for (int i = head, j; i != tail; i = next[i]) {
47                 if (graph[head][next[i]] && graph[tail][i]) {
48                     for (j = head; j != i; j = next[j]) {
49                         last[next[j]] = j;
50                     }
51                     j = next[head];
52                     next[head] = next[i];
53                     next[tail] = i;
54                     tail = j;

```

```
55         for (j = i; j != head; j = last[j]) {
56             next[j] = last[j];
57         }
58         break;
59     }
60 }
61 }
62 next[tail] = head;
63 if (right[0] > n) {
64     break;
65 }
66 for (int i = head; i != tail; i = next[i]) {
67     if (adjacent(i)) {
68         head = next[i];
69         tail = i;
70         next[tail] = 0;
71         break;
72     }
73 }
74 }
75 std::vector<int> answer;
76 for (int i = head; ; i = next[i]) {
77     if (i == 1) {
78         answer.push_back(i);
79         for (int j = next[i]; j != i; j = next[j]) {
80             answer.push_back(j);
81         }
82         answer.push_back(i);
83         break;
84     }
85     if (i == tail) {
86         break;
87     }
88 }
89 return answer;
90 }
```

Chapter 5

字符串

5.1 模式串匹配

```
1 void build(char *pattern) {
2     int length = (int)strlen(pattern + 1);
3     fail[0] = -1;
4     for (int i = 1, j; i <= length; ++i) {
5         for (j = fail[i - 1]; j != -1 && pattern[i] != pattern[j + 1]; j = fail[j]);
6         fail[i] = j + 1;
7     }
8 }
9
10 void solve(char *text, char *pattern) {
11     int length = (int)strlen(text + 1);
12     for (int i = 1, j; i <= length; ++i) {
13         for (j = match[i - 1]; j != -1 && text[i] != pattern[j + 1]; j = fail[j]);
14         match[i] = j + 1;
15     }
16 }
```

5.2 坚固的模式串匹配

```
1 lenA = strlen(A); lenB = strlen(B);
2 nxt[0] = lenB, nxt[1] = lenB - 1;
3 for (int i = 0; i <= lenB; i++)
4     if (B[i] != B[i + 1]) {nxt[1] = i; break;}
5 int j, k = 1, p, L;
6 for (int i = 2; i < lenB; i++) {
7     p = k + nxt[k] - 1; L = nxt[i - k];
8     if (i + L <= p) nxt[i] = L;
9     else {
10         j = p - i + 1;
11         if (j < 0) j = 0;
12         while (i + j < lenB && B[i + j] == B[j]) j++;
13         nxt[i] = j; k = i;
```

```

14     }
15 }
16 int minlen = lenA <= lenB ? lenA : lenB; ex[0] = minlen;
17 for (int i = 0; i < minlen; i++)
18     if (A[i] != B[i]) {ex[0] = i; break;}
19 k = 0;
20 for (int i = 1; i < lenA; i++){
21     p = k + ex[k] - 1; L = next[i - k];
22     if (i + L <= p) ex[i] = L;
23     else {
24         j = p - i + 1;
25         if (j < 0) j = 0;
26         while (i + j < lenA && j < lenB && A[i + j] == B[j]) j++;
27         ex[i] = j; k = i;
28     }
29 }

```

5.3 AC 自动机

```

1  int size, c[MAXT][26], f[MAXT], fail[MAXT], d[MAXT];
2
3  int alloc() {
4      size++;
5      std::fill(c[size], c[size] + 26, 0);
6      f[size] = fail[size] = d[size] = 0;
7      return size;
8  }
9
10 void insert(char *s) {
11     int len = strlen(s + 1), p = 1;
12     for (int i = 1; i <= len; i++) {
13         if (c[p][s[i] - 'a']) p = c[p][s[i] - 'a'];
14         else{
15             int newnode = alloc();
16             c[p][s[i] - 'a'] = newnode;
17             d[newnode] = s[i] - 'a';
18             f[newnode] = p;
19             p = newnode;
20         }
21     }
22 }
23
24 void buildfail() {
25     static int q[MAXT];
26     int left = 0, right = 0;
27     fail[1] = 0;
28     for (int i = 0; i < 26; i++) {
29         c[0][i] = 1;
30         if (c[1][i]) q[++right] = c[1][i];
31     }
32     while (left < right) {

```

```

33     left++;
34     int p = fail[f[q[left]]];
35     while (!c[p][d[q[left]]]) p = fail[p];
36     fail[q[left]] = c[p][d[q[left]]];
37     for (int i = 0; i < 26; i++) {
38         if (c[q[left]][i]) {
39             q[++right] = c[q[left]][i];
40         }
41     }
42 }
43 for (int i = 1; i <= size; i++)
44     for (int j = 0; j < 26; j++) {
45         int p = i;
46         while (!c[p][j]) p = fail[p];
47         c[i][j] = c[p][j];
48     }
49 }

```

5.4 后缀数组

```

1 namespace suffix_array{
2     int wa[MAXN], wb[MAXN], ws[MAXN], wv[MAXN];
3     bool cmp(int *r, int a, int b, int l) {
4         return r[a] == r[b] && r[a + l] == r[b + l];
5     }
6     void DA(int *r, int *sa, int n, int m) {
7         int *x = wa, *y = wb, *t;
8         for (int i = 0; i < m; i++) ws[i] = 0;
9         for (int i = 0; i < n; i++) ws[x[i]]++;
10        for (int i = 1; i < m; i++) ws[i] += ws[i - 1];
11        for (int i = n - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
12        for (int i, j = 1, p = 1; p < n; j <= 1, m = p) {
13            for (p = 0, i = n - j; i < n; i++) y[p++] = i;
14            for (i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
15            for (i = 0; i < n; i++) wv[i] = x[y[i]];
16            for (i = 0; i < m; i++) ws[i] = 0;
17            for (i = 0; i < n; i++) ws[wv[i]]++;
18            for (i = 1; i < m; i++) ws[i] += ws[i - 1];
19            for (i = n - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
20            for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
21                x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
22        }
23    }
24    void getheight(int *r, int *sa, int *rk, int *h, int n) {
25        for (int i = 1; i <= n; i++) rk[sa[i]] = i;
26        for (int i = 0, j, k = 0; i < n; h[rk[i++]] = k)
27            for (k ? k-- : 0, j = sa[rk[i] - 1]; r[i + k] == r[j + k]; k++);
28    }
29 };

```

5.5 广义后缀自动机

```

1 // Generalized Suffix Automaton
2 void add(int x, int &last) {
3     int lastnode = last;
4     if (c[lastnode][x]) {
5         int nownode = c[lastnode][x];
6         if (l[nownode] == l[lastnode] + 1) last = nownode;
7         else{
8             int auxnode = ++size; l[auxnode] = l[lastnode] + 1;
9             for (int i = 0; i < 26; i++) c[auxnode][i] = c[nownode][i];
10            f[auxnode] = f[nownode]; f[nownode] = auxnode;
11            for (; lastnode && c[lastnode][x] == nownode; lastnode = f[lastnode]) {
12                c[lastnode][x] = auxnode;
13            }
14            last = auxnode;
15        }
16    }
17    else{
18        int newnode = ++size; l[newnode] = l[lastnode] + 1;
19        for (; lastnode && !c[lastnode][x]; lastnode = f[lastnode]) c[lastnode][x] = newnode;
20        if (!lastnode) f[newnode] = 1;
21        else{
22            int nownode = c[lastnode][x];
23            if (l[lastnode] + 1 == l[nownode]) f[newnode] = nownode;
24            else{
25                int auxnode = ++size; l[auxnode] = l[lastnode] + 1;
26                for (int i = 0; i < 26; i++) c[auxnode][i] = c[nownode][i];
27                f[auxnode] = f[nownode]; f[nownode] = f[newnode] = auxnode;
28                for (; lastnode && c[lastnode][x] == nownode; lastnode = f[lastnode]) {
29                    c[lastnode][x] = auxnode;
30                }
31            }
32        }
33        last = newnode;
34    }
35 }

```

5.6 Manacher 算法

```

1 void manacher(char *text, int length) {
2     palindrome[0] = 1;
3     for (int i = 1, j = 0; i < length; ++i) {
4         if (j + palindrome[j] <= i) {
5             palindrome[i] = 0;
6         } else {
7             palindrome[i] = std::min(palindrome[(j << 1) - i], j + palindrome[j] - i);
8         }
9         while (i - palindrome[i] >= 0 && i + palindrome[i] < length
10             && text[i - palindrome[i]] == text[i + palindrome[i]]) {

```



```

11         palindrome[i]++;
12     }
13     if (i + palindrome[i] > j + palindrome[j]) {
14         j = i;
15     }
16 }
17 }

```

5.7 回文树

```

1  struct Palindromic_Tree{
2      int nTree, nStr, last, c[MAXT][26], fail[MAXT], r[MAXN], l[MAXN], s[MAXN];
3      int allocate(int len) {
4          l[nTree] = len;
5          r[nTree] = 0;
6          fail[nTree] = 0;
7          memset(c[nTree], 0, sizeof(c[nTree]));
8          return nTree++;
9      }
10     void init() {
11         nTree = nStr = 0;
12         int newEven = allocate(0);
13         int newOdd = allocate(-1);
14         last = newEven;
15         fail[newEven] = newOdd;
16         fail[newOdd] = newEven;
17         s[0] = -1;
18     }
19     void add(int x) {
20         s[++nStr] = x;
21         int nownode = last;
22         while (s[nStr - l[nownode] - 1] != s[nStr]) nownode = fail[nownode];
23         if (!c[nownode][x]) {
24             int newnode = allocate(l[nownode] + 2), &newfail = fail[newnode];
25             newfail = fail[nownode];
26             while (s[nStr - l[newfail] - 1] != s[nStr]) newfail = fail[newfail];
27             newfail = c[newfail][x];
28             c[nownode][x] = newnode;
29         }
30         last = c[nownode][x];
31         r[last]++;
32     }
33     void count() {
34         for (int i = nTree - 1; i >= 0; i--) {
35             r[fail[i]] += r[i];
36         }
37     }
38 }

```

5.8 循环串最小表示

```
1 int solve(char *text, int length) {
2     int i = 0, j = 1, delta = 0;
3     while (i < length && j < length && delta < length) {
4         char tokeni = text[(i + delta) % length];
5         char tokenj = text[(j + delta) % length];
6         if (tokeni == tokenj) {
7             delta++;
8         } else {
9             if (tokeni > tokenj) {
10                i += delta + 1;
11            } else {
12                j += delta + 1;
13            }
14            if (i == j) {
15                j++;
16            }
17            delta = 0;
18        }
19    }
20    return std::min(i, j);
21 }
```

Chapter 6

计算几何

6.1 二维基础

6.1.1 点类

```
1 struct Point{
2     double x, y;
3     Point() {}
4     Point(double x, double y):x(x), y(y) {}
5     Point operator +(const Point &p) const {
6         return Point(x + p.x, y + p.y);
7     }
8     Point operator -(const Point &p) const {
9         return Point(x - p.x, y - p.y);
10    }
11    Point operator *(const double &p) const {
12        return Point(x * p, y * p);
13    }
14    Point operator /(const double &p) const {
15        return Point(x / p, y / p);
16    }
17    int read() {
18        return scanf("%lf%lf", &x, &y);
19    }
20 };
21
22 struct Line{
23     Point a, b;
24     Line() {}
25     Line(Point a, Point b):a(a), b(b) {}
26 };
```

6.1.2 凸包

```
1 bool Pair_Comp(const Point &a, const Point &b) {
2     if (dcmp(a.x - b.x) < 0) return true;
```

```

3     if (dcmp(a.x - b.x) > 0) return false;
4     return dcmp(a.y - b.y) < 0;
5 }
6
7 int Convex_Hull(int n, Point *P, Point *C) {
8     sort(P, P + n, Pair_Comp);
9     int top = 0;
10    for (int i = 0; i < n; i++) {
11        while (top >= 2 && dcmp(det(C[top - 1] - C[top - 2], P[i] - C[top - 2])) <= 0) top--;
12        C[top++] = P[i];
13    }
14    int lasttop = top;
15    for (int i = n - 1; i >= 0; i--) {
16        while (top > lasttop && dcmp(det(C[top - 1] - C[top - 2], P[i] - C[top - 2])) <= 0)
17            top--;
18        C[top++] = P[i];
19    }
20    return top;
}

```

6.1.3 半平面交

```

1 bool isOnLeft(const Point &x, const Line &l) {
2     double d = det(x - l.a, l.b - l.a);
3     return dcmp(d) <= 0;
4 }
5 // 传入一个线段的集合L, 传出A, 并且返回A的大小
6 int getIntersectionOfHalfPlane(int n, Line *L, Line *A) {
7     Line *q = new Line[n + 1];
8     Point *p = new Point[n + 1];
9     sort(L, L + n, Polar_Angle_Comp_Line);
10    int l = 1, r = 0;
11    for (int i = 0; i < n; i++) {
12        while (l < r && !isOnLeft(p[r - 1], L[i])) r--;
13        while (l < r && !isOnLeft(p[l], L[i])) l++;
14        q[++r] = L[i];
15        if (l < r && is_Colinear(q[r], q[r - 1])) {
16            r--;
17            if (isOnLeft(L[i].a, q[r])) q[r] = L[i];
18        }
19        if (l < r) p[r - 1] = getIntersection(q[r - 1], q[r]);
20    }
21    while (l < r && !isOnLeft(p[r - 1], q[l])) r--;
22    if (r - l + 1 <= 2) return 0;
23    int tot = 0;
24    for (int i = l; i <= r; i++) A[tot++] = q[i];
25    return tot;
26 }

```

6.1.4 最近点对

```

1  bool comparex(const Point &a, const Point &b) {
2      return sgn(a.x - b.x) < 0;
3  }
4
5  bool comparey(const Point &a, const Point &b) {
6      return sgn(a.y - b.y) < 0;
7  }
8
9  double solve(const std::vector<Point> &point, int left, int right) {
10     if (left == right) {
11         return INF;
12     }
13     if (left + 1 == right) {
14         return dist(point[left], point[right]);
15     }
16     int mid = left + right >> 1;
17     double result = std::min(solve(left, mid), solve(mid + 1, right));
18     std::vector<Point> candidate;
19     for (int i = left; i <= right; ++i) {
20         if (std::abs(point[i].x - point[mid].x) <= result) {
21             candidate.push_back(point[i]);
22         }
23     }
24     std::sort(candidate.begin(), candidate.end(), comparey);
25     for (int i = 0; i < (int)candidate.size(); ++i) {
26         for (int j = i + 1; j < (int)candidate.size(); ++j) {
27             if (std::abs(candidate[i].y - candidate[j].y) >= result) {
28                 break;
29             }
30             result = std::min(result, dist(candidate[i], candidate[j]));
31         }
32     }
33     return result;
34 }
35
36 double solve(std::vector<Point> point) {
37     std::sort(point.begin(), point.end(), comparex);
38     return solve(point, 0, (int)point.size() - 1);
39 }

```

6.2 三维基础

6.2.1 点类

```

1  int dcmp(const double &x) {
2      return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1);
3  }
4
5  struct TPoint{
6      double x, y, z;
7      TPoint() {}

```

```

8   TPoint(double x, double y, double z) : x(x), y(y), z(z) {}
9   TPoint operator +(const TPoint &p) const {
10      return TPoint(x + p.x, y + p.y, z + p.z);
11  }
12  TPoint operator -(const TPoint &p) const {
13      return TPoint(x - p.x, y - p.y, z - p.z);
14  }
15  TPoint operator *(const double &p) const {
16      return TPoint(x * p, y * p, z * p);
17  }
18  TPoint operator /(const double &p) const {
19      return TPoint(x / p, y / p, z / p);
20  }
21  bool operator <(const TPoint &p) const {
22      int dX = dcmp(x - p.x), dY = dcmp(y - p.y), dZ = dcmp(z - p.z);
23      return dX < 0 || (dX == 0 && (dY < 0 || (dY == 0 && dZ < 0)));
24  }
25  bool read() {
26      return scanf("%lf%lf%lf", &x, &y, &z) == 3;
27  }
28 };
29
30 double sqrdist(const TPoint &a) {
31     double ret = 0;
32     ret += a.x * a.x;
33     ret += a.y * a.y;
34     ret += a.z * a.z;
35     return ret;
36 }
37 double sqrdist(const TPoint &a, const TPoint &b) {
38     double ret = 0;
39     ret += (a.x - b.x) * (a.x - b.x);
40     ret += (a.y - b.y) * (a.y - b.y);
41     ret += (a.z - b.z) * (a.z - b.z);
42     return ret;
43 }
44 double dist(const TPoint &a) {
45     return sqrt(sqrdist(a));
46 }
47 double dist(const TPoint &a, const TPoint &b) {
48     return sqrt(sqrdist(a, b));
49 }
50 TPoint det(const TPoint &a, const TPoint &b) {
51     TPoint ret;
52     ret.x = a.y * b.z - b.y * a.z;
53     ret.y = a.z * b.x - b.z * a.x;
54     ret.z = a.x * b.y - b.x * a.y;
55     return ret;
56 }
57 double dot(const TPoint &a, const TPoint &b) {
58     double ret = 0;
59     ret += a.x * b.x;
60     ret += a.y * b.y;

```

```

61     ret += a.z * b.z;
62     return ret;
63 }
64 double detdot(const TPoint &a, const TPoint &b, const TPoint &c, const TPoint &d) {
65     return dot(det(b - a, c - a), d - a);
66 }

```

6.2.2 凸包

```

1  struct Triangle{
2      TPoint a, b, c;
3      Triangle() {}
4      Triangle(TPoint a, TPoint b, TPoint c) : a(a), b(b), c(c) {}
5      double getArea() {
6          TPoint ret = det(b - a, c - a);
7          return dist(ret) / 2.0;
8      }
9  };
10 namespace Convex_Hull {
11     struct Face{
12         int a, b, c;
13         bool isOnConvex;
14         Face() {}
15         Face(int a, int b, int c) : a(a), b(b), c(c) {}
16     };
17
18     int nFace, left, right, whe[MAXN][MAXN];
19     Face queue[MAXF], tmp[MAXF];
20
21     bool isVisible(const std::vector<TPoint> &p, const Face &f, const TPoint &a) {
22         return dcmp(detdot(p[f.a], p[f.b], p[f.c], a)) > 0;
23     }
24
25     bool init(std::vector<TPoint> &p) {
26         bool check = false;
27         for (int i = 1; i < (int)p.size(); i++) {
28             if (dcmp(sqrdist(p[0], p[i])) {
29                 std::swap(p[1], p[i]);
30                 check = true;
31                 break;
32             }
33         }
34         if (!check) return false;
35         check = false;
36         for (int i = 2; i < (int)p.size(); i++) {
37             if (dcmp(sqrdist(det(p[i] - p[0], p[1] - p[0])))) {
38                 std::swap(p[2], p[i]);
39                 check = true;
40                 break;
41             }
42         }
43         if (!check) return false;

```

```

44     check = false;
45     for (int i = 3; i < (int)p.size(); i++) {
46         if (dcmp(detdot(p[0], p[1], p[2], p[i]))) {
47             std::swap(p[3], p[i]);
48             check = true;
49             break;
50         }
51     }
52     if (!check) return false;
53     for (int i = 0; i < (int)p.size(); i++)
54         for (int j = 0; j < (int)p.size(); j++) {
55             whe[i][j] = -1;
56         }
57     return true;
58 }
59
60 void pushface(const int &a, const int &b, const int &c) {
61     nFace++;
62     tmp[nFace] = Face(a, b, c);
63     tmp[nFace].isOnConvex = true;
64     whe[a][b] = nFace;
65     whe[b][c] = nFace;
66     whe[c][a] = nFace;
67 }
68
69 bool deal(const std::vector<TPoint> &p, const std::pair<int, int> &now, const TPoint &base)
70 {
71     int id = whe[now.second][now.first];
72     if (!tmp[id].isOnConvex) return true;
73     if (isVisible(p, tmp[id], base)) {
74         queue[++right] = tmp[id];
75         tmp[id].isOnConvex = false;
76         return true;
77     }
78     return false;
79 }
80
81 std::vector<Triangle> getConvex(std::vector<TPoint> &p) {
82     static std::vector<Triangle> ret;
83     ret.clear();
84     if (!init(p)) return ret;
85     if (!isVisible(p, Face(0, 1, 2), p[3])) pushface(0, 1, 2); else pushface(0, 2, 1);
86     if (!isVisible(p, Face(0, 1, 3), p[2])) pushface(0, 1, 3); else pushface(0, 3, 1);
87     if (!isVisible(p, Face(0, 2, 3), p[1])) pushface(0, 2, 3); else pushface(0, 3, 2);
88     if (!isVisible(p, Face(1, 2, 3), p[0])) pushface(1, 2, 3); else pushface(1, 3, 2);
89     for (int a = 4; a < (int)p.size(); a++) {
90         TPoint base = p[a];
91         for (int i = 1; i <= nFace; i++) {
92             if (tmp[i].isOnConvex && isVisible(p, tmp[i], base)) {
93                 left = 0, right = 0;
94                 queue[++right] = tmp[i];
95                 tmp[i].isOnConvex = false;
96                 while (left < right) {

```



```

96         Face now = queue[++left];
97         if (!deal(p, std::make_pair(now.a, now.b), base)) pushface(now.a, now.
98             b, a);
99         if (!deal(p, std::make_pair(now.b, now.c), base)) pushface(now.b, now.
100             c, a);
101         if (!deal(p, std::make_pair(now.c, now.a), base)) pushface(now.c, now.
102             a, a);
103     }
104     break;
105 }
106 }
107 for (int i = 1; i <= nFace; i++) {
108     Face now = tmp[i];
109     if (now.isOnConvex) {
110         ret.push_back(Triangle(p[now.a], p[now.b], p[now.c]));
111     }
112 }
113 return ret;
114 }
115 };
116 int n;
117 std::vector<TPoint> p;
118 std::vector<Triangle> answer;
119 int main() {
120     scanf("%d", &n);
121     for (int i = 1; i <= n; i++) {
122         TPoint a;
123         a.read();
124         p.push_back(a);
125     }
126     answer = Convex_Hull::getConvex(p);
127     double areaCounter = 0.0;
128     for (int i = 0; i < (int)answer.size(); i++) {
129         areaCounter += answer[i].getArea();
130     }
131     printf("%.3f\n", areaCounter);
132     return 0;
133 }

```

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

```

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

```

```

8  }
9  Matrix getRotate(const double &a, const double &b, const double &c, const double &theta) {
10     Matrix ret;
11     ret.a[0][0] = a * a * (1 - cos(theta)) + cos(theta);
12     ret.a[0][1] = a * b * (1 - cos(theta)) + c * sin(theta);
13     ret.a[0][2] = a * c * (1 - cos(theta)) - b * sin(theta);
14     ret.a[0][3] = 0;
15
16     ret.a[1][0] = b * a * (1 - cos(theta)) - c * sin(theta);
17     ret.a[1][1] = b * b * (1 - cos(theta)) + cos(theta);
18     ret.a[1][2] = b * c * (1 - cos(theta)) + a * sin(theta);
19     ret.a[1][3] = 0;
20
21     ret.a[2][0] = c * a * (1 - cos(theta)) + b * sin(theta);
22     ret.a[2][1] = c * b * (1 - cos(theta)) - a * sin(theta);
23     ret.a[2][2] = c * c * (1 - cos(theta)) + cos(theta);
24     ret.a[2][3] = 0;
25
26     ret.a[3][0] = 0;
27     ret.a[3][1] = 0;
28     ret.a[3][2] = 0;
29     ret.a[3][3] = 1;
30     return ret;
31 }
32 Matrix getRotate(const double &ax, const double &ay, const double &az, const double &bx, const
    double &by, const double &bz, const double &theta) {
33     double l = dist(Point(0, 0, 0), Point(bx, by, bz));
34     Matrix ret = getTrans(-ax, -ay, -az);
35     ret = ret * getRotate(bx / l, by / l, bz / l, theta);
36     ret = ret * getTrans(ax, ay, az);
37     return ret;
38 }

```

6.3 多边形

6.3.1 判断点在多边形内部

```

1  bool point_on_line(const Point &p, const Point &a, const Point &b) {
2      return sgn(det(p, a, b)) == 0 && sgn(dot(p, a, b)) <= 0;
3  }
4
5  bool point_in_polygon(const Point &p, const std::vector<Point> &polygon) {
6      int counter = 0;
7      for (int i = 0; i < (int)polygon.size(); ++i) {
8          Point a = polygon[i], b = polygon[(i + 1) % (int)polygon.size()];
9          if (point_on_line(p, a, b)) {
10             // Point on the boundary are excluded.
11             return false;
12         }
13         int x = sgn(det(a, p, b));
14         int y = sgn(a.y - p.y);

```

```

15         int z = sgn(b.y - p.y);
16         counter += (x > 0 && y <= 0 && z > 0);
17         counter -= (x < 0 && z <= 0 && y > 0);
18     }
19     return counter;
20 }

```

6.3.2 多边形内整点计数

```

1  int getInside(int n, Point *P) { // 求多边形P内有多少个整数点
2      int OnEdge = n;
3      double area = getArea(n, P);
4      for (int i = 0; i < n - 1; i++) {
5          Point now = P[i + 1] - P[i];
6          int y = (int)now.y, x = (int)now.x;
7          OnEdge += abs(gcd(x, y)) - 1;
8      }
9      Point now = P[0] - P[n - 1];
10     int y = (int)now.y, x = (int)now.x;
11     OnEdge += abs(gcd(x, y)) - 1;
12     double ret = area - (double)OnEdge / 2 + 1;
13     return (int)ret;
14 }

```

6.4 圆

6.4.1 最小覆盖圆

```

1  Point getmid(Point a, Point b) {
2      return Point((a.x + b.x) / 2, (a.y + b.y) / 2);
3  }
4  Point getcross(Point a, Point vA, Point b, Point vB) {
5      Point u = a - b;
6      double t = det(vB, u) / det(vA, vB);
7      return a + vA * t;
8  }
9  Point getcir(Point a, Point b, Point c) {
10     Point midA = getmid(a, b), vA = Point(-(b - a).y, (b - a).x);
11     Point midB = getmid(b, c), vB = Point(-(c - b).y, (c - b).x);
12     return getcross(midA, vA, midB, vB);
13 }
14 double mincir(Point *p, int n) {
15     std::random_shuffle(p + 1, p + n + 1);
16     Point O = p[1];
17     double r = 0;
18     for (int i = 2; i <= n; i++) {
19         if (dist(O, p[i]) <= r) continue;
20         O = p[i]; r = 0;
21         for (int j = 1; j < i; j++) {
22             if (dist(O, p[j]) <= r) continue;

```

```

23         O = getmid(p[i], p[j]); r = dist(O, p[i]);
24         for (int k = 1; k < j; k++) {
25             if (dist(O, p[k]) <= r) continue;
26             O = getcir(p[i], p[j], p[k]);
27             r = dist(O, p[i]);
28         }
29     }
30 }
31 return r;
32 }

```

6.4.2 最小覆盖球

```

1  double eps(1e-8);
2  int sign(const double & x) {
3      return (x > eps) - (x + eps < 0);
4  }
5  bool equal(const double & x, const double & y) {
6      return x + eps > y and y + eps > x;
7  }
8  struct Point {
9      double x, y, z;
10     Point() {
11     }
12     Point(const double & x, const double & y, const double & z) : x(x), y(y), z(z) {
13     }
14     void scan() {
15         scanf("%lf%lf%lf", &x, &y, &z);
16     }
17     double sqrlen() const {
18         return x * x + y * y + z * z;
19     }
20     double len() const {
21         return sqrt(sqrlen());
22     }
23     void print() const {
24         printf("(%lf_ %lf_ %lf)\n", x, y, z);
25     }
26 } a[33];
27 Point operator + (const Point & a, const Point & b) {
28     return Point(a.x + b.x, a.y + b.y, a.z + b.z);
29 }
30 Point operator - (const Point & a, const Point & b) {
31     return Point(a.x - b.x, a.y - b.y, a.z - b.z);
32 }
33 Point operator * (const double & x, const Point & a) {
34     return Point(x * a.x, x * a.y, x * a.z);
35 }
36 double operator % (const Point & a, const Point & b) {
37     return a.x * b.x + a.y * b.y + a.z * b.z;
38 }
39 Point operator * (const Point & a, const Point & b) {

```

```

40     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);
41 }
42 struct Circle {
43     double r;
44     Point o;
45     Circle() {
46         o.x = o.y = o.z = r = 0;
47     }
48     Circle(const Point & o, const double & r) : o(o), r(r) {
49     }
50     void scan() {
51         o.scan();
52         scanf("%lf", &r);
53     }
54     void print() const {
55         o.print();
56         printf("%lf\n", r);
57     }
58 };
59 struct Plane {
60     Point nor;
61     double m;
62     Plane(const Point & nor, const Point & a) : nor(nor){
63         m = nor % a;
64     }
65 };
66 Point intersect(const Plane & a, const Plane & b, const Plane & c) {
67     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);
68     return 1 / ((c1 * c2) % c3) * Point((c4 * c2) % c3, (c1 * c4) % c3, (c1 * c2) % c4);
69 }
70 bool in(const Point & a, const Circle & b) {
71     return sign((a - b.o).len() - b.r) <= 0;
72 }
73 bool operator < (const Point & a, const Point & b) {
74     if(!equal(a.x, b.x)) {
75         return a.x < b.x;
76     }
77     if(!equal(a.y, b.y)) {
78         return a.y < b.y;
79     }
80     if(!equal(a.z, b.z)) {
81         return a.z < b.z;
82     }
83     return false;
84 }
85 bool operator == (const Point & a, const Point & b) {
86     return equal(a.x, b.x) and equal(a.y, b.y) and equal(a.z, b.z);
87 }
88 vector<Point> vec;
89 Circle calc() {
90     if(vec.empty()) {
91         return Circle(Point(0, 0, 0), 0);

```

```

92     }else if(1 == (int)vec.size()) {
93         return Circle(vec[0], 0);
94     }else if(2 == (int)vec.size()) {
95         return Circle(0.5 * (vec[0] + vec[1]), 0.5 * (vec[0] - vec[1]).len());
96     }else if(3 == (int)vec.size()) {
97         double r((vec[0] - vec[1]).len() * (vec[1] - vec[2]).len() * (vec[2] - vec[0]).len() /
98                 2 / fabs(((vec[0] - vec[2]) * (vec[1] - vec[2])).len()));
99         return Circle(intersect(Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0])),
100                                Plane(vec[2] - vec[1], 0.5 * (vec[2] + vec[1])),
101                                Plane((vec[1] - vec[0]) * (vec[2] - vec[0]), vec[0])), r);
102     }else {
103         Point o(intersect(Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0])),
104                           Plane(vec[2] - vec[0], 0.5 * (vec[2] + vec[0])),
105                           Plane(vec[3] - vec[0], 0.5 * (vec[3] + vec[0]))));
106         return Circle(o, (o - vec[0]).len());
107     }
108 Circle miniBall(int n) {
109     Circle res(calc());
110     for(int i(0); i < n; i++) {
111         if(!in(a[i], res)) {
112             vec.push_back(a[i]);
113             res = miniBall(i);
114             vec.pop_back();
115             if(i) {
116                 Point tmp(a[i]);
117                 memmove(a + 1, a, sizeof(Point) * i);
118                 a[0] = tmp;
119             }
120         }
121     }
122     return res;
123 }
124 int main() {
125     int n;
126     for(;;) {
127         scanf("%d", &n);
128         if(!n) {
129             break;
130         }
131         for(int i(0); i < n; i++) {
132             a[i].scan();
133         }
134         sort(a, a + n);
135         n = unique(a, a + n) - a;
136         vec.clear();
137         printf("%.10f\n", miniBall(n).r);
138     }

```

6.4.3 多边形与圆的交面积

```
1 // 求扇形面积
```

```

2  double getSectorArea(const Point &a, const Point &b, const double &r) {
3      double c = (2.0 * r * r - sqrdist(a, b)) / (2.0 * r * r);
4      double alpha = acos(c);
5      return r * r * alpha / 2.0;
6  }
7  // 求二次方程 $ax^2 + bx + c = 0$ 的解
8  std::pair<double, double> getSolution(const double &a, const double &b, const double &c) {
9      double delta = b * b - 4.0 * a * c;
10     if (dcmp(delta) < 0) return std::make_pair(0, 0);
11     else return std::make_pair((-b - sqrt(delta)) / (2.0 * a), (-b + sqrt(delta)) / (2.0 * a))
12     ;
13 }
14 // 直线与圆的交点
15 std::pair<Point, Point> getIntersection(const Point &a, const Point &b, const double &r) {
16     Point d = b - a;
17     double A = dot(d, d);
18     double B = 2.0 * dot(d, a);
19     double C = dot(a, a) - r * r;
20     std::pair<double, double> s = getSolution(A, B, C);
21     return std::make_pair(a + d * s.first, a + d * s.second);
22 }
23 // 原点到线段AB的距离
24 double getPointDist(const Point &a, const Point &b) {
25     Point d = b - a;
26     int sA = dcmp(dot(a, d)), sB = dcmp(dot(b, d));
27     if (sA * sB <= 0) return det(a, b) / dist(a, b);
28     else return std::min(dist(a), dist(b));
29 }
30 // a和b和原点组成的三角形与半径为r的圆的交的面积
31 double getArea(const Point &a, const Point &b, const double &r) {
32     double dA = dot(a, a), dB = dot(b, b), dC = getPointDist(a, b), ans = 0.0;
33     if (dcmp(dA - r * r) <= 0 && dcmp(dB - r * r) <= 0) return det(a, b) / 2.0;
34     Point tA = a / dist(a) * r;
35     Point tB = b / dist(b) * r;
36     if (dcmp(dC - r) > 0) return getSectorArea(tA, tB, r);
37     std::pair<Point, Point> ret = getIntersection(a, b, r);
38     if (dcmp(dA - r * r) > 0 && dcmp(dB - r * r) > 0) {
39         ans += getSectorArea(tA, ret.first, r);
40         ans += det(ret.first, ret.second) / 2.0;
41         ans += getSectorArea(ret.second, tB, r);
42         return ans;
43     }
44     if (dcmp(dA - r * r) > 0) return det(ret.first, b) / 2.0 + getSectorArea(tA, ret.first, r)
45     ;
46     else return det(a, ret.second) / 2.0 + getSectorArea(ret.second, tB, r);
47 }
48 // 求圆与多边形的交的主过程
49 double getArea(int n, Point *p, const Point &c, const double r) {
50     double ret = 0.0;
51     for (int i = 0; i < n; i++) {
52         int sgn = dcmp(det(p[i] - c, p[(i + 1) % n] - c));
53         if (sgn > 0) ret += getArea(p[i] - c, p[(i + 1) % n] - c, r);
54         else ret -= getArea(p[(i + 1) % n] - c, p[i] - c, r);
55     }
56 }

```

```
53     }  
54     return fabs(ret);  
55 }
```


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 巴什博弈

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 $n = m + 1r + s$, (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:

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

7.2.3 阶梯博弈

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

```

1  import java.io.*;
2  import java.util.*;
3  import java.math.*;
4
5  public class Main {
6      static int get(char c) {
7          if (c <= '9')
8              return c - '0';
9          else if (c <= 'Z')
10             return c - 'A' + 10;
11         else
12             return c - 'a' + 36;
13     }
14     static char get(int x) {
15         if (x <= 9)
16             return (char) (x + '0');
17         else if (x <= 35)
18             return (char) (x - 10 + 'A');
19         else
20             return (char) (x - 36 + 'a');
21     }
22     static BigInteger get(String s, BigInteger x) {

```

```

23     BigInteger ans = BigInteger.valueOf(0), now = BigInteger.valueOf(1);
24     for (int i = s.length() - 1; i >= 0; i--) {
25         ans = ans.add(now.multiply(BigInteger.valueOf(get(s.charAt(i)))));
26         now = now.multiply(x);
27     }
28     return ans;
29 }
30 public static void main(String [] args) {
31     Scanner cin = new Scanner(new BufferedInputStream(System.in));
32     for (; ; ) {
33         BigInteger x = cin.nextBigInteger();
34         if (x.compareTo(BigInteger.valueOf(0)) == 0)
35             break;
36         String s = cin.next(), t = cin.next(), r = "";
37         BigInteger ans = get(s, x).mod(get(t, x));
38         if (ans.compareTo(BigInteger.valueOf(0)) == 0)
39             r = "0";
40         for (; ans.compareTo(BigInteger.valueOf(0)) > 0;) {
41             r = get(ans.mod(x).intValue()) + r;
42             ans = ans.divide(x);
43         }
44         System.out.println(r);
45     }
46 }
47 }
48
49 // Arrays
50 int a[];
51 .fill(a[, int fromIndex, int toIndex],val); | .sort(a[, int fromIndex, int toIndex])
52 // String
53 String s;
54 .charAt(int i); | compareTo(String) | compareToIgnoreCase () | contains(String) |
55 length () | substring(int l, int len)
56 // BigInteger
57 .abs() | .add() | bitLength () | subtract () | divide () | remainder () | divideAndRemainder
58   () | modPow(b, c) |
59 pow(int) | multiply () | compareTo () |
60 gcd() | intValue () | longValue () | isProbablePrime(int c) (1 - 1/2^c) |
61 nextProbablePrime () | shiftLeft(int) | valueOf ()
62 // BigDecimal
63 .ROUND_CEILING | ROUND_DOWN_FLOOR | ROUND_HALF_DOWN | ROUND_HALF_EVEN | ROUND_HALF_UP |
64   ROUND_UP
65 .divide(BigDecimal b, int scale , int round_mode) | doubleValue () | movePointLeft(int) | pow(
66   int) |
67 setScale(int scale , int round_mode) | stripTrailingZeros ()
68 // StringBuilder
69 StringBuilder sb = new StringBuilder ();
70 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. $fib_m | fib_n^2 \Leftrightarrow n fib_n | m$

8.1.3 错排公式

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

8.1.4 莫比乌斯函数

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

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

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

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

8.1.5 Burnside 引理

设 G 是一个有限群, 作用在集合 X 上. 对每个 g 属于 G , 令 X^g 表示 X 中在 g 作用下的不动元素, 轨道数 (记作 $|X/G|$) 由如下公式给出:

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

8.1.6 五边形数定理

设 $p(n)$ 是 n 的拆分数, 有

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

8.1.7 树的计数

1. 有根树计数: $n+1$ 个结点的有根树的个数为

$$a_{n+1} = \frac{\sum_{j=1}^n j \cdot a_j \cdot S_{n,j}}{n}$$

其中,

$$S_{n,j} = \sum_{i=1}^{n/j} a_{n+1-ij} = S_{n-j,j} + a_{n+1-j}$$

2. 无根树计数: 当 n 为奇数时, n 个结点的无根树的个数为

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

当 n 为偶数时, n 个结点的无根树的个数为

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

3. n 个结点的完全图的生成树个数为

$$n^{n-2}$$

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

8.1.8 欧拉公式

平面图的顶点个数、边数和面的个数有如下关系:

$$V - E + F = C + 1$$

其中, V 是顶点的数目, E 是边的数目, F 是面的数目, C 是组成图形的连通部分的数目。当图是单连通图的时候, 公式简化为:

$$V - E + F = 2$$

8.1.9 皮克定理

给定顶点坐标均是整点 (或正方形格点) 的简单多边形, 其面积 A 和内部格点数目 i 、边上格点数目 b 的关系:

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

8.1.10 牛顿恒等式

设

$$\prod_{i=1}^n (x - x_i) = a_n + a_{n-1}x + \cdots + a_1x^{n-1} + a_0x^n$$

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

则

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

特别地, 对于

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

有

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

8.2 平面几何公式

8.2.1 三角形

1. 半周长

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

2. 面积

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

3. 中线

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

4. 角平分线

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

5. 高线

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

6. 内切圆半径

$$\begin{aligned} r &= \frac{S}{p} = \frac{\arcsin \frac{B}{2} \cdot \sin \frac{C}{2}}{\sin \frac{B+C}{2}} = 4R \cdot \sin \frac{A}{2} \sin \frac{B}{2} \sin \frac{C}{2} \\ &= \sqrt{\frac{(p-a)(p-b)(p-c)}{p}} = p \cdot \tan \frac{A}{2} \tan \frac{B}{2} \tan \frac{C}{2} \end{aligned}$$

7. 外接圆半径

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

8.2.2 四边形

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

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

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

3. 对于圆内接四边形

$$ac + bd = D_1 D_2$$

4. 对于圆内接四边形

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

8.2.3 正 n 边形

R 为外接圆半径, r 为内切圆半径

1. 中心角

$$A = \frac{2\pi}{n}$$

2. 内角

$$C = \frac{n-2}{n}\pi$$

3. 边长

$$a = 2\sqrt{R^2 - r^2} = 2R \cdot \sin \frac{A}{2} = 2r \cdot \tan \frac{A}{2}$$

4. 面积

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

8.2.4 圆

1. 弧长

$$l = rA$$

2. 弦长

$$a = 2\sqrt{2hr - h^2} = 2r \cdot \sin \frac{A}{2}$$

3. 弓形高

$$h = r - \sqrt{r^2 - \frac{a^2}{4}} = r(1 - \cos \frac{A}{2}) = \frac{1}{2} \cdot \arctan \frac{A}{4}$$

4. 扇形面积

$$S_1 = \frac{rl}{2} = \frac{r^2 A}{2}$$

5. 弓形面积

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

8.2.5 棱柱

1. 体积

$$V = Ah$$

A 为底面积, h 为高

2. 侧面积

$$S = lp$$

l 为棱长, p 为直截面周长

3. 全面积

$$T = S + 2A$$

8.2.6 棱锥

1. 体积

$$V = Ah$$

A 为底面积, h 为高

2. 正棱锥侧面积

$$S = lp$$

l 为棱长, p 为直截面周长

3. 正棱锥全面积

$$T = S + 2A$$

8.2.7 棱台

1. 体积

$$V = (A_1 + A_2 + \sqrt{A_1 A_2}) \cdot \frac{h}{3}$$

A_1, A_2 为上下底面积, h 为高

2. 正棱台侧面积

$$S = \frac{p_1 + p_2}{2} l$$

p_1, p_2 为上下底面周长, l 为斜高

3. 正棱台全面积

$$T = S + A_1 + A_2$$

8.2.8 圆柱

1. 侧面积

$$S = 2\pi r h$$

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

$$S = \pi r l$$

3. 全面积

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

4. 体积

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

8.2.10 圆台

1. 母线

$$l = \sqrt{h^2 + (r_1 - r_2)^2}$$

2. 侧面积

$$S = \pi(r_1 + r_2)l$$

3. 全面积

$$T = \pi r_1(l + r_1) + \pi r_2(l + r_2)$$

4. 体积

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

8.2.11 球

1. 全面积

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

2. 体积

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

8.2.12 球台

1. 侧面积

$$S = 2\pi r h$$

2. 全面积

$$T = \pi(2rh + r_1^2 + r_2^2)$$

3. 体积

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

8.2.13 球扇形

1. 全面积

$$T = \pi r(2h + r_0)$$

h 为球冠高, r_0 为球冠底面半径

2. 体积

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

8.3 立体几何公式**8.3.1 球面三角公式**

设 a, b, c 是边长, A, B, C 是所对的二面角, 有余弦定理

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

正弦定理

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

三角形面积是 $A + B + C - \pi$

8.3.2 四面体体积公式

U, V, W, u, v, w 是四面体的 6 条棱, U, V, W 构成三角形, $(U, u), (V, v), (W, w)$ 互为对棱, 则

$$V = \frac{\sqrt{(s-2a)(s-2b)(s-2c)(s-2d)}}{192uvw}$$

其中

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