
STANDARD CODE LIBRARY
OF
HUST AFFILIATED KINDERGARTEN

VERSION 1.2

EDITED BY

SDDYZJH

DRAGOONKILLER

DREACTOR

Huazhong University of Science and Technology

目录

计算几何	4
几何通用	4
平面几何通用	6
立体几何通用	8
判断点在凸多边形内	10
凸包	11
旋转卡壳	12
最小覆盖圆	13
数据结构	14
KD 树	14
Splay	16
表达式解析	21
并查	25
可持久化线段树	26
轻重边剖分	28
手写 bitset	31
树状数组	33
线段树	34
左偏树	36
动态规划	39
插头 DP	39
概率 DP	41
数位 DP	42
四边形 DP	43
斜率 DP	45
状压 DP	46
最长上升子序列	48
图论	49
best's therom	49
k 短路可持久化堆	51
spfa 费用流	54
Tarjan 有向图强连通分量	56
zkw 费用流	57

倍增 LCA	59
点分治	59
堆优化 dijkstra	64
矩阵树定理	65
平面欧几里得距离最小生成树	68
最大流 Dinic	75
最大团	78
最小度限制生成树	80
最优比率生成树	84
数学	85
常用公式	85
积性函数	85
莫比乌斯反演	86
常用等式	86
SG 函数	86
矩阵乘法快速幂	87
线性基	88
线性筛	90
整数卷积 NTT	91
中国剩余定理	92
字符串	93
AC 自动机	93
KMP	95
Manacher	96
Trie 树	97
后缀数组-DC3	98
后缀数组-倍增法	99
后缀自动机	100
扩展 KMP	102
杂项	103
测速	103
日期公式	104
读入挂	104
高精度	105

康托展开与逆展开	110
快速乘	111
模拟退火	111
常用概念	112
映射	112
反演	112
弦图	113
五边形数	113
重心	113
第二类 Bernoulli number	113
Catalan 数	113
Stirling 数	113
三角公式	114

计算几何

几何通用

```
1  /// 计算几何专用. 按需选用.
2
3  db eps = 1e-12; // 线性误差范围; long double : 1e-16;
4  db eps2 = 1e-6; // 平方级误差范围; long double: 1e-8;
5  bool eq(db a, db b) { return abs(a-b) < eps; }
6
7  /// ===== 点和向量 =====
8  struct point;
9  struct point
10 {
11     db x, y;
12     point():x(0),y(0) { }
13     point(db a,db b):x(a),y(b) { }
14     point(point const& f):x(f.x),y(f.y) { }
15     point operator=(point const& f) { x=f.x; y=f.y; return *this; }
16
17     point operator+(point const& b) const { return point(x + b.x, y + b.y);
18         }
19     point operator-(point const& b) const { return point(x - b.x, y - b.y);
20         }
21     point operator()(point const& b) const { return b - *this; } // 从本顶点
22         出发,指向另一个点的向量.
23
24     db len2() const { return x*x+y*y; } // 模的平方.
25     db len() const { return sqrt(len2()); } // 向量的模.
26     point norm() const { db l = len(); return point(x/l, y/l); } // 标准化.
27
28     /// 把向量旋转f个弧度.
29     point rot(double const& f) const
30     { return point(x*cos(f) - y*sin(f), x*sin(f) + y*cos(f)); }
31
32     /// 极角, +x轴为0, 弧度制, (- , ].
33     db pangle() const { if(y >= 0) return acos(x / len()); else return -
34         acos(x / len()); }
35
36     void out() const { printf("%.2f, %.2f", x, y); } // 输出.
37 };
38
39 /// 数乘.
40 point operator*(point const& a, db const& b) { return point(a.x * b, a.y * b
    ); }
41 point operator*(db const& b, point const& a) { return point(a.x * b, a.y * b
    ); }
42
43 /// 叉积.
44 db operator*(point const& a, point const& b) { return a.x * b.y - a.y * b.x;
```

```

    }
41 // 点积.
42 db operator&(point const& a, point const& b) { return a.x * b.x + a.y * b.y;
    }
43
44 bool operator==(point const& a, point const& b) { return eq(a.x, b.x) && eq(
    a.y, b.y); }
45
46 // 判断本向量在另一个向量的顺时针方向. 注意选用eps或0.
47 bool operator>>(point const& a, point const& b) { return a*b > eps; }
48 // 判断本向量在另一个向量的顺时针方向或同向. 注意选用eps或0.
49 bool operator>=(point const& a, point const& b) { return a*b > -eps; }
50
51 // ===== 线段 =====
52 struct segment
53 {
54     point from,to;
55     segment(point const& a = point(), point const& b = point()) : from(a),
        to(b) { }
56
57     point dir() const { return to - from; } // 方向向量,未标准化.
58
59     db len() const { return dir().len(); } // 长度.
60
61     // 点在线段上.
62     bool overlap(point const& v) const
63     { return eq(from(to).len(), v(from).len() + v(to).len()); }
64
65     point projection(point const& p) const // 点到直线上的投影.
66     {
67         db h = abs(dir() * from(p)) / len();
68         db r = sqrt(from(p).len2() - h*h);
69         if(eq(r, 0)) return from;
70         if((from(to) & from(p)) < 0) return from(to).norm() * (-r);
71         else return from(to).norm() * r;
72     }
73
74     point nearest(point const& p) const // 点到线段的最近点.
75     {
76         point g = projection(p);
77         if(overlap(g)) return g;
78         if(g(from).len() < g(to).len()) return from;
79         return to;
80     }
81 };
82
83 bool operator/(segment const& a, segment const& b) // 平行 (零向量平行于任意
    向量).
84 {
85     return eq(a.dir() * b.dir(), 0);
86 }

```

```

87
88 // 相交. 不计线段端点则删掉 eq(..., 0) 的所有判断.
89 bool operator*(segment const& A, segment const& B)
90 {
91     point dia = A.from(A.to);
92     point dib = B.from(B.to);
93     db a = dia * A.from(B.from);
94     db b = dia * A.from(B.to);
95     db c = dib * B.from(A.from);
96     db d = dib * B.from(A.to);
97     return ((a < 0 && b > 0) || (a > 0 && b < 0) || A.overlap(B.from) || A.
          overlap(B.to)) &&
98         ((c < 0 && d > 0) || (c > 0 && d < 0) || B.overlap(A.from) || B.
          overlap(A.to));
99 }

```

平面几何通用

```

1  /// 计算几何专用. 按需选用.
2
3  db eps = 1e-12; // 线性误差范围; long double : 1e-16;
4  db eps2 = 1e-6; // 平方级误差范围; long double: 1e-8;
5  bool eq(db a, db b) { return abs(a-b) < eps; }
6
7  // ===== 点和向量 =====
8  struct point;
9  struct point
10 {
11     db x, y;
12     point():x(0),y(0) { }
13     point(db a,db b):x(a),y(b) { }
14     point(point const& f):x(f.x),y(f.y) { }
15     point operator=(point const& f) { x=f.x; y=f.y; return *this; }
16
17     point operator+(point const& b) const { return point(x + b.x, y + b.y);
18     }
19     point operator-(point const& b) const { return point(x - b.x, y - b.y);
20     }
21     point operator()(point const& b) const { return b - *this; } // 从本顶点
22     出发,指向另一个点的向量.
23
24     db len2() const { return x*x+y*y; } // 模的平方.
25     db len() const { return sqrt(len2()); } // 向量的模.
26     point norm() const { db l = len(); return point(x/l, y/l); } // 标准化.
27
28     // 把向量旋转f个弧度.
29     point rot(double const& f) const
30     { return point(x*cos(f) - y*sin(f), x*sin(f) + y*cos(f)); }

```

```

29 // 极角, +x轴为0, 弧度制, (-, ].
30 db pangle() const { if(y >= 0) return acos(x / len()); else return -
    acos(x / len()); }
31
32 void out() const { printf("%.2f, %.2f", x, y); } // 输出.
33 };
34
35 // 数乘.
36 point operator*(point const& a, db const& b) { return point(a.x * b, a.y * b
    ); }
37 point operator*(db const& b, point const& a) { return point(a.x * b, a.y * b
    ); }
38
39 // 叉积.
40 db operator*(point const& a, point const& b) { return a.x * b.y - a.y * b.x;
    }
41 // 点积.
42 db operator*(point const& a, point const& b) { return a.x * b.x + a.y * b.y;
    }
43
44 bool operator==(point const& a, point const& b) { return eq(a.x, b.x) && eq(
    a.y, b.y); }
45
46 // 判断本向量在另一个向量的顺时针方向. 注意选用eps或0.
47 bool operator>>(point const& a, point const& b) { return a*b > eps; }
48 // 判断本向量在另一个向量的顺时针方向或同向. 注意选用eps或0.
49 bool operator>=(point const& a, point const& b) { return a*b > -eps; }
50
51 // ===== 线段 =====
52 struct segment
53 {
54     point from, to;
55     segment(point const& a = point(), point const& b = point()) : from(a),
        to(b) { }
56
57     point dir() const { return to - from; } // 方向向量, 未标准化.
58
59     db len() const { return dir().len(); } // 长度.
60
61     // 点在线段上.
62     bool overlap(point const& v) const
63     { return eq(from.to).len(), v(from).len() + v(to).len()); }
64
65     point projection(point const& p) const // 点到直线上的投影.
66     {
67         db h = abs(dir() * from(p)) / len();
68         db r = sqrt(from(p).len2() - h*h);
69         if(eq(r, 0)) return from;
70         if((from(to) & from(p)) < 0) return from + from(to).norm() * (-r);
71         else return from + from(to).norm() * r;
72     }

```



```

73
74     point nearest(point const& p) const // 点到线段的最近点.
75     {
76         point g = projection(p);
77         if(overlap(g)) return g;
78         if(g(from).len() < g(to).len()) return from;
79         return to;
80     }
81 };
82
83 bool operator/(segment const& a, segment const& b) // 平行 (零向量平行于任意
84     向量).
85 {
86     return eq(a.dir() * b.dir(), 0);
87 }
88 // 相交. 不计线段端点则删掉 eq(..., 0) 的所有判断.
89 bool operator*(segment const& A, segment const& B)
90 {
91     point dia = A.from(A.to);
92     point dib = B.from(B.to);
93     db a = dia * A.from(B.from);
94     db b = dia * A.from(B.to);
95     db c = dib * B.from(A.from);
96     db d = dib * B.from(A.to);
97     return ((a < 0 && b > 0) || (a > 0 && b < 0) || A.overlap(B.from) || A.
98         overlap(B.to)) &&
99         ((c < 0 && d > 0) || (c > 0 && d < 0) || B.overlap(A.from) || B.
100         overlap(A.to));

```

立体几何通用

```

1 db eps = 1e-12; // 线性误差范围; long double : 1e-16;
2 db eps2 = 1e-6; // 平方级误差范围; long double: 1e-8;
3 bool eq(db a, db b) { return abs(a-b) < eps; }
4
5 // ===== 点和向量 =====
6 struct point;
7 struct point
8 {
9     db x, y, z;
10     point():x(0),y(0),z(0) { }
11     point(db a,db b,db c):x(a),y(b),z(c) { }
12     point(point const& f):x(f.x),y(f.y),z(f.z) { }
13     point operator=(point const& f) { x=f.x; y=f.y; z=f.z; return *this; }
14
15     point operator+(point const& b) const { return point(x + b.x, y + b.y, z
        + b.z); }

```

```

16     point operator-(point const& b) const { return point(x - b.x, y - b.y, z
17         - b.z); }
18
19     db len2() const { return x*x+y*y+z*z; } // 模的平方.
20     db len() const { return sqrt(len2()); } // 向量的模.
21     point norm() const { db l = len(); return point(x/l, y/l, z/l); } // 标
22         准化.
23
24     void out(const char* c) const { printf("%.2f,□%.2f,□%.2f)%s", x, y, z,
25         c); } // 输出.
26
27 // 数乘.
28 point operator*(point const& a, db const& b) { return point(a.x * b, a.y * b
29     , a.z * b); }
30 point operator*(db const& b, point const& a) { return point(a.x * b, a.y * b
31     , a.z * b); }
32
33 // 叉积.
34 point operator*(point const& a, point const& b)
35 { 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); }
36
37 // 点积.
38 db operator*(point const& a, point const& b)
39 { return a.x * b.x + a.y * b.y + a.z * b.z; }
40
41 bool operator==(point const& a, point const& b)
42 { return eq(a.x, b.x) && eq(a.y, b.y) && eq(a.z, b.z); }
43
44 // ===== 线段 =====
45 struct segment
46 {
47     point from,to;
48     segment() : from(), to() { }
49     segment(point const& a, point const& b) : from(a), to(b) { }
50
51     point dir() const { return to - from; } // 方向向量,未标准化.
52     db len() const { return dir().len(); } // 长度.
53     db len2() const { return dir().len2(); }
54
55     // 点在线段上.
56     bool overlap(point const& v) const
57     { return eq(from(to).len(), v(from).len() + v(to).len()); }
58
59     point projection(point const& p) const // 点到直线上的投影.
60     {
61         db h2 = abs((dir() * from(p)).len2()) / len2();
62         db r = sqrt(from(p).len2() - h2);

```

```

61         if(eq(r, 0)) return from;
62         if((from(to) & from(p)) < 0) return from + from(to).norm() * (-r);
63         else return from + from(to).norm() * r;
64     }
65
66     point nearest(point const& p) const // 点到线段的最近点.
67     {
68         point g = projection(p);
69         if(overlap(g)) return g;
70         if(g(from).len() < g(to).len()) return from;
71         return to;
72     }
73
74     point nearest(segment const& x) const // 线段x上的离本线段最近的点.
75     {
76         db l = 0.0, r = 1.0;
77         while(r - l > eps)
78         {
79             db delta = r - l;
80             db lmid = l + 0.4 * delta;
81             db rmid = l + 0.6 * delta;
82             point lp = x.interpolate(lmid);
83             point rp = x.interpolate(rmid);
84             point lnear = nearest(lp);
85             point rnear = nearest(rp);
86             if(lp(lnear).len2() > rp(rnear).len2()) l = lmid;
87             else r = rmid;
88         }
89         return x.interpolate(l);
90     }
91
92     point interpolate(db const& p) const { return from + p * dir(); }
93 };
94
95 bool operator/(segment const& a, segment const& b) // 平行 (零向量平行于任意
96     向量).
97 {
98     return eq((a.dir() * b.dir()).len(), 0);
99 }

```

判断点在凸多边形内

```

1  /// 在线，单次询问O(logn)，st为凸包点数，包括多边形上顶点和边界.
2  /// 要求凸包上没有相同点，仅包含顶点.
3
4  bool agcmp(point const& a, point const& b) { return sp(a) * sp(b) < 0; }
5  bool PointInside(point target)
6  {
7      sp = stk[0];

```

```

8     point vt = sp(stk[1]);
9     point vb = sp(stk[st-2]);
10    db mt = vt * sp(target);
11    db mb = vb * sp(target);
12    bool able = (eq(mt, 0) && eq(mb, 0)) ||
13                (eq(mt, 0) && mb > 0) || (eq(mb, 0) && mt < 0) ||
14                (mt < 0 && mb > 0);
15    if(able)
16    {
17        int xp = (int)(lower_bound(stk+1, stk+st-2, target, agcmp) - stk);
18        able &= !(segment(sp, target) * segment(stk[xp], stk[xp-1]));
19        able |= segment(stk[xp], stk[xp-1]).overlap(target);
20    }
21    return able;
22 }
23
24 /// 在线, 单次询问O(logn), st为凸包点数, **不**包括多边形上顶点和边界.
25
26 bool agcmp(point const& a, point const& b) { return sp(a) * sp(b) < 0; }
27 bool PointInside(point target)
28 {
29     sp = stk[0];
30     point vt = sp(stk[1]);
31     point vb = sp(stk[st-2]);
32     db mt = vt * sp(target);
33     db mb = vb * sp(target);
34     bool able = mt < 0 && mb > 0;
35     if(able)
36     {
37         int xp = (int)(lower_bound(stk+1, stk+st-2, target, agcmp) - stk);
38         able &= !(segment(sp, target) * segment(stk[xp], stk[xp-1]));
39     }
40     return able;
41 }

```

凸包

```

1  /// 凸包
2  /// 去除输入中重复顶点, 保留头尾重复, 顺时针顺序.
3
4  /// a: 输入点.
5  /// stk: 用来存凸包上的点的栈.
6  /// st: 栈顶下标, 指向最后一个元素的下一个位置.
7  /// stk[0]: 凸包上 y 值最小的点中, x 值最小的点.
8
9  //////////////////////////////////////
10
11 int n;
12 point a[105000];

```

```

13 point stk[105000]; int st;
14
15 bool operator<(point const& a, point const& b) { return eq(a.y, b.y) ? a.x <
    b.x : a.y < b.y; }
16 // 使用 >> 则取凸包上的点.
17 // 使用 >>= 不取凸包上的点.
18 void Graham()
19 {
20     sort(a, a+n);
21     int g = (int)(unique(a, a+n) - a);
22     st=0;
23
24     for(int i=0; i<g; i++)
25     {
26         while(st>1 && (stk[st-2](stk[st-1]) >> stk[st-1](a[i]))) st--;
27         stk[st++]=a[i];
28     }
29     int p=st;
30     for(int i=g-2; i>=0; i--)
31     {
32         while(st>p && (stk[st-2](stk[st-1]) >> stk[st-1](a[i]))) st--;
33         stk[st++]=a[i];
34     }
35 }
36
37 /// [...] AC HDU 1392

```

旋转卡壳

```

1
2 /// 旋转卡壳求最远点对距离.
3 /// stk[]: 按顺序存储的凸壳上的点的数组.
4 //////////////////////////////////////
5 int GetmaxDistance()
6 {
7     int res=0;
8     int p=2;
9     for(int i=1; i<st; i++)
10    {
11        while( p!=st && area(stk[i-1], stk[i], stk[p+1]) > area(stk[i-1],
            stk[i], stk[p]))
12            p++;
13        // 此时stk[i]的对踵点是stk[p].
14        if(p==st) break;
15        // 修改至想要的部分.
16        res=max(res, stk[i-1](stk[p]).dist2());
17        res=max(res, stk[i](stk[p]).dist2());
18    }
19    return res;

```

```
20 }
```

最小覆盖圆

```
1  /// 最小覆盖圆.
2
3  /// n: 点数.
4  /// a: 输入点的数组.
5
6  //////////////////////////////////////
7
8  const db eps = 1e-12;
9  const db eps2 = 1e-8;
10
11  /// 过三点的圆的圆心.
12  point CC(point const& a, point const& b, point const& c)
13  {
14      point ret;
15      db a1 = b.x-a.x, b1 = b.y-a.y, c1 = (a1*a1+b1*b1)*0.5;
16      db a2 = c.x-a.x, b2 = c.y-a.y, c2 = (a2*a2+b2*b2)*0.5;
17      db d = a1*b2 - a2*b1;
18      if(abs(d)<eps) return (b+c)*0.5;
19      ret.x=a.x+(c1*b2-c2*b1)/d;
20      ret.y=a.y+(a1*c2-a2*c1)/d;
21      return ret;
22  }
23
24  int n;
25  point a[1005000];
26
27  struct Resault{ db x,y,r; };
28  Resault MCC()
29  {
30      if(n==0) return {0, 0, 0};
31      if(n==1) return {a[0].x, a[0].y, 0};
32      if(n==2) return {(a[0]+a[1]).x*0.5, (a[0]+a[1]).y*0.5, dist(a[0],a[1])
33                      *0.5};
34
35
36      for(int i=0;i<n;i++) swap(a[i], a[rand()%n]); // 随机交换.
37
38      point O; db R = 0.0;
39      for(int i=2; i<n; i++) if(O(a[i]).len() >= R+eps2)
40      {
41          O=a[i];
42          R=0.0;
43          for(int j=0; j<i; j++) if(O(a[j]).len() >= R+eps2)
44          {
45              O=(a[i] + a[j]) * 0.5;
```

```

45         R=a[i](a[j]).len() * 0.5;
46
47         for(int k=0; k<j; k++) if(O(a[k]).len() >= R+eps2)
48         {
49             O = CC(a[i], a[j], a[k]);
50             R = O(a[i]).len();
51         }
52     }
53 }
54
55 return {O.x, O.y, R};
56 }

```

数据结构

KD 树

```

1
2 /// KD 树.
3 /// 最近邻点查询.
4 /// 维度越少剪枝优化效率越高. 4 维时是1/10倍运行时间, 8维时是1/3倍运行时间.
5 /// 板子使用欧几里得距离.
6 /// 可以把距离修改成曼哈顿距离之类的, **剪枝一般不会出错**.
7
8 //////////////////////////////////////
9
10 const int mxnc = 105000; // 最大的所有树节点数总量.
11 const int dem = 4; // 维度数量.
12
13 const db INF = 1e20;
14
15 /// 空间中的点.
16 struct point
17 {
18     db v[dem]; // 维度坐标.
19     // 注意你有可能用到每个维度坐标是不同的*类型*的点.
20     // 此时需要写两个点对于第k个维度坐标的比较函数.
21     point() { }
22     point(db* coord) { memcpy(v, coord, sizeof(v)); }
23     point(point const& x) { memcpy(v, x.v, sizeof(v)); }
24
25     point& operator=(point const& x)
26     { memcpy(v, x.v, sizeof(v)); return *this; }
27
28     db& operator[] (int const& k) { return v[k]; }
29     db const& operator[] (int const& k) const { return v[k]; }
30 };
31

```

```

32 db dist(point const& x, point const& y)
33 {
34     db a = 0.0;
35     for(int i=0; i<dem; i++) a += (x[i] - y[i]) * (x[i] - y[i]);
36     return sqrt(a);
37 }
38
39 /// 树中的节点.
40 struct node
41 {
42     point loc; // 节点坐标点.
43     int d; // 该节点的下层节点从哪个维度切割. 切割坐标值由该节点坐标值
44             给出.
45     node* s[2]; // 左右子节点.
46
47     int sep(point const& x) const { return x[d] >= loc[d]; }
48 };
49 node pool[mxnc]; node* curn = pool;
50
51 /// 这个数组用来分配唯独特切割顺序. 可以改用别的维度选择方式.
52 int flc[] = {3, 0, 2, 1};
53 node* newnode(point const& p, int dep)
54 {
55     curn->loc = p;
56     curn->d = flc[dep % dem];
57     curn->s[0] = curn->s[1] = NULL;
58     return curn++;
59 }
60
61 /// KD树.
62 struct KDTree
63 {
64     node* root;
65
66     KDTree() { root = NULL; }
67
68     node* insert(point const& x)
69     {
70         node* cf = NULL;
71         node* cur = root;
72         int dep = 0;
73         while(cur != NULL)
74         {
75             dep++;
76             cf = cur;
77             cur = cur->s[cur->sep(x)];
78         }
79         if(cf == NULL) return root = newnode(x, dep);
80         return cf->s[cf->sep(x)] = newnode(x, dep);
81     }

```



```

82 // 求最近点的距离，以及最近点。
83 pair<db, point*> nearest(point const& p, node* x)
84 {
85     if(x == NULL) return make_pair(INF, (point*)NULL);
86
87     int k = x->sep(p);
88
89     // 拿到点 p 从属子区域的结果。
90     pair<db, point*> sol = nearest(p, x->s[k]);
91
92     // 用当前区域存储的点更新答案。
93     db cd = dist(x->loc, p);
94     if(sol.first > cd)
95     {
96         sol.first = cd;
97         sol.second = &(x->loc);
98     }
99
100    // 如果当前结果半径和另一个子区域相交，询问子区域并更新答案。
101    db divDist = abs(p[x->d] - x->loc[x->d]);
102    if(sol.first >= divDist)
103    {
104        pair<db, point*> solx = nearest(p, x->s[!k]);
105        if(sol.first > solx.first) sol = solx;
106    }
107
108    return sol;
109 }
110
111 db nearestDist(point const& p) { return nearest(p, root).first; }
112 };
113
114 /// 初始化节点列表，会清除**所有树**的信息。
115 void Init()
116 {
117     cur = pool;
118 }

```

Splay

```

1 /// Splay.
2 /// 没有特殊功能的平衡树。预留了一个自底向上更新的update函数。
3 /// pool: 点的池子。Splay数据结构本身只保存根节点指针。
4 /// 重新初始化: nt = pool + 1; 不要更改nil.
5
6 /// mxn: 节点池子大小。
7 const int mxn = 205000;
8
9 //////////////////////////////////////

```

```

10
11 struct node* nil;
12 struct node
13 {
14     int v;
15     int cnt;
16     node*s[2];
17     node*f;
18     void update()
19     {
20         cnt=1;
21         if(s[0]!=nil) cnt+=s[0]->cnt;
22         if(s[1]!=nil) cnt+=s[1]->cnt;
23     }
24 }
25 pool[mxn]; node* nt=pool;
26
27 node*newnode(int v, node*f)
28 {
29     nt->v=v;
30     nt->cnt=1;
31     nt->s[0]=nt->s[1]=nil;
32     nt->f=f;
33     return nt++;
34 }
35
36
37 struct SplayTree
38 {
39     node*root;
40     SplayTree():root(nil){}
41
42     void rot(node*x)
43     {
44         node*y=x->f;
45         int k=(x==y->s[0]);
46
47         y->s[k^1]=x->s[k];
48         if(x->s[k]!=nil) x->s[k]->f=y;
49
50         x->f=y->f;
51         if(y->f!=nil) y->f->s[y==y->f->s[1]]=x;
52
53         y->f=x; x->s[k]=y;
54
55         y->update();
56     }
57
58     node* splay(node*x,node*t=nil)
59     {
60         while(x->f!=t)

```

```

61     {
62         node*y=x->f;
63         if (y->f!=t)
64             if ((x==y->s[0])^(y==y->f->s[0]))
65                 rot(x); else rot(y);
66         rot(x);
67     }
68     x->update();
69     if (t==nil) root=x;
70     return x;
71 }
72
73 //=====
74
75 void Insert(int v)
76 {
77     if (root==nil) { root=newnode(v, nil); return; }
78     node *x=root, *y=root;
79     while (x!=nil) { y=x; x=x->s[x->v <= v]; }
80     splay(y->s[y->v<=v] = newnode(v, y));
81 }
82
83
84 node*Find(int v) // 查找值相等的节点. 找不到会返回nil.
85 {
86     node *x=root, *y=root;
87     node *r=nil;
88     while (x!=nil)
89     {
90         y=x;
91         if (x->v==v) r=x;
92         x=x->s[x->v < v];
93     }
94     splay(y);
95     return r;
96 }
97
98 node* FindRank(int k) // 查找排名为 k 的节点.
99 {
100     node *x=root, *y=root;
101     while (x!=nil)
102     {
103         y=x;
104         if (k==x->s[0]->cnt+1) break;
105         if (k<x->s[0]->cnt+1) x=x->s[0];
106         else { k=x->s[0]->cnt+1; x=x->s[1]; }
107     }
108     splay(y);
109     return x;
110 }
111

```

```

112 // 排名从 1 开始.
113 int GetRank(node*x) { return splay(x)->s[0]->cnt+1; }
114
115 node*Delete(node*x)
116 {
117     int k=GetRank(x);
118     node*L=FindRank(k-1);
119     node*R=FindRank(k+1);
120
121     if(L!=nil) splay(L);
122     if(R!=nil) splay(R,L);
123
124     if(L==nil && R==nil) root=nil;
125     else if(R==nil) L->s[1]=nil;
126     else R->s[0]=nil;
127
128     if(R!=nil) R->update();
129     if(L!=nil) L->update();
130
131     return x;
132 }
133
134 node*Prefix(int v) // 前驱.
135 {
136     node *x=root, *y=root;
137     node*r=nil;
138     while(x!=nil)
139     {
140         y=x;
141         if(x->v<v) r=x;
142         x=x->s[x->v<v];
143     }
144     splay(y);
145     return r;
146 }
147
148 node*Suffix(int v) // 后继.
149 {
150     node *x=root, *y=root;
151     node*r=nil;
152     while(x!=nil)
153     {
154         y=x;
155         if(x->v>v) r=x;
156         x=x->s[x->v>v];
157     }
158     splay(y);
159     return r;
160 }
161
162 //=====

```

```

163 void output() { output(root); printf("%s\n",root==nil ? "empty_tree!" :
    ""); }
164 void output(node*x)
165 {
166     if(x==nil)return ;
167     output(x->s[0]);
168     printf("%d\n",x->v);
169     output(x->s[1]);
170 }
171
172 void test() { test(root); printf("%s\n",root==nil ? "empty_tree!" : "");
    }
173 void test(node*x)
174 {
175     if(x==nil)return ;
176     test(x->s[0]);
177     printf("%pL[%d]:%dL:%pR:%pCnt:%d\n",x,x->v,x->f,x->s[0],x
        ->s[1],x->cnt);
178     test(x->s[1]);
179 }
180
181 };
182
183
184 int n;
185
186 int main()
187 {
188     nil=newnode(-1, nullptr);
189     nil->cnt=0;
190     nil->f=nil->s[0]=nil->s[1]=nil;
191
192     n=getint();
193     SplayTree st;
194
195     for(int i=0;i<n;i++)
196     {
197         int c;
198         c=getint();
199         switch(c)
200         {
201             case 1: //Insert
202                 c=getint();
203                 st.Insert(c);
204             break;
205             case 2: //Delete
206                 c=getint();
207                 st.Delete(st.Find(c));
208             break;
209             case 3: //Rank
210                 c=getint();

```

```

211         printf("%d\n", st.GetRank(st.Find(c)));
212         break;
213         case 4: //FindRank
214             c=getint();
215             printf("%d\n", st.FindRank(c)->v);
216             break;
217         case 5: //prefix
218             c=getint();
219             printf("%d\n", st.Prefix(c)->v);
220             break;
221         case 6: //suffix
222             c=getint();
223             printf("%d\n", st.Suffix(c)->v);
224             break;
225         case 7: //test
226             st.test();
227             break;
228         default: break;
229     }
230 }
231
232 return 0;
233 }

```

表达式解析

```

1  /// 表达式解析
2  /// 线性扫描，直接计算。
3  /// 不支持三元运算符。
4  /// 一元运算符经过特殊处理。它们不会(也不应)与二元运算符共用一种符号。
5
6  /// prio: 字符优先级。在没有括号的约束下，优先级高的优先计算。
7  /// pref: 结合顺序。pref[i] == true 表示从左到右结合，false 则为从右到左结合。
8
9  /// 圆括号运算符会特别对待。
10
11 /// 如果需要建树，直接改Calc和Push函数。
12
13 /// ctt: 字符集编号下界。
14 /// ctf: 字符集编号上界。
15 /// ctx: 字符集大小。
16 const int ctf = -128;
17 const int ctt = 127;
18 const int ctx = ctt - ctf;
19
20 /// 表达式字符总数。
21 const int mxn = 1005000;
22
23 /// inp: 输入的表达式；已经去掉了空格。

```

```

23  /// inpt: 输入的表达式长度.
24  /// sx, aval: 由Destruct设定的外部变量数组. 无需改动.
25  /// 用法:
26  int len = Destruct(inp, inpt);
27  Evaluate(sx, len, aval);
28
29
30  /// 重新初始化: 调用Destruct即可.
31
32  //////////////////////////////////////
33
34  int __prio[ctx]; int* prio = __prio - ctf;
35  bool __pref[ctx]; bool* pref = __pref - ctf;
36
37  // 设置一个运算符的优先级和结合顺序.
38  void SetProp(char x, int a, int b) { prio[x] = a; pref[x] = b; }
39
40  stack<int> ap; // 变量栈.
41  stack<char> op; // 符号栈.
42
43  int Fetch() { int x = ap.top(); ap.pop(); return x; }
44  void Push(int x) { ap.push(x); }
45
46  /// 这个函数定义了如何处理栈内的实际元素.
47  void Calc()
48  {
49      char cop = op.top(); op.pop();
50      switch(cop)
51      {
52          case '+': { int b = Fetch(); int a = Fetch(); Push(a + b); } return;
53          case '-': { int b = Fetch(); int a = Fetch(); Push(a - b); } return;
54          case '*': { int b = Fetch(); int a = Fetch(); Push(a * b); } return;
55          case '/': { int b = Fetch(); int a = Fetch(); Push(a / b); } return;
56          case '|': { int b = Fetch(); int a = Fetch(); Push(a | b); } return;
57          case '&': { int b = Fetch(); int a = Fetch(); Push(a & b); } return;
58          case '^': { int b = Fetch(); int a = Fetch(); Push(a ^ b); } return;
59          case '!': { int a = Fetch(); Push(a); } return; // '+'的一元算符
60
61          case '~': { int a = Fetch(); Push(-a); } return; // '-'的一元算符
62
63          default: return;
64      }
65  }
66
67  /// s: 转化后的表达式, 其中0表示变量, 其它表示相应运算符. len: 表达式长度.
68  /// g: 变量索引序列, 表示表达式从左到右的变量分别是哪个.
69  void Evaluate(char* s, int len, int* g)
70  {
71      int gc = 0;
72      for(int i=0; i<len; i++)
73      {

```

```

72         if(s[i] == 0) // 输入是一个变量. 一般可以直接按需求改掉, 例如 if(
           IsVar(s[i])).
73     {
74         Push(g[gc++]); // 第gc个变量的**值**入栈.
75     }
76     else // 输入是一个运算符s[i].
77     {
78         if(s[i] == '(') op.push(s[i]);
79         else if(s[i] == ')')
80         {
81             while(op.top() != '(') Calc();
82             op.pop();
83         }
84         else
85         {
86             while( prio[s[i]] < prio[op.top()] ||
87                   (prio[s[i]] == prio[op.top()] && pref[s[i]] == true))
88                 Calc();
89             op.push(s[i]);
90         }
91     }
92 }
93 }
94
95 /// 解析一个字符串, 得到能够被上面的函数处理的格式.
96 /// 对于这个函数而言, "变量"是某个十进制整数.
97 /// 有些时候输入本身就是这样的格式, 就不需要过多处理.
98 /// 支持的二元运算符: +, -, *, /, |, &, ^. 支持的一元运算符: +, -.
99 char sx[mxn]; // 表达式序列.
100 int aval[mxn]; // 数字. 这些是扔到变量栈里面的东西.
101 // 可以直接写成某种place holder, 如果不关心这些变量之间的区别
    的话.
102 /// 返回: 表达式序列长度.
103 int Destruct(char* s, int len)
104 {
105     int xlen = 0;
106     sx[xlen++] = '(';
107     bool cvr = false;
108     int x = 0;
109     int vt = 0;
110     for(int i=0; i<len; i++)
111     {
112         if('0' <= s[i] && s[i] <= '9')
113         {
114             if(!cvr) sx[xlen++] = 0;
115             cvr = true;
116             if(cvr) x = x * 10 + s[i] - '0';
117         }
118         else
119         {
120             if(cvr) { aval[vt++] = x; x = 0; }

```



```

121         cvr = false;
122         sx[xlen++] = s[i];
123     }
124 }
125 if(cvr) { aval[vt++] = x; x = 0; }
126
127 for(int i=xlen; i>=1; i--) // 一元运算符特判, 修改成不同于二元运算符的符号.
128     if((sx[i]=='+' || sx[i]=='-') && sx[i-1] != ')') && sx[i-1])
129         sx[i] = sx[i] == '+' ? '!' : '~';
130
131 sx[xlen++] = ')';
132 return xlen;
133 }
134
135 char c[mxn];
136
137 char inp[mxn]; int inpt;
138 int main()
139 {
140     SetProp('(', 0, true);
141     SetProp(')', 0, true);
142
143     SetProp('+', 10, true);
144     SetProp('-', 10, true);
145
146     SetProp('*', 100, true);
147     SetProp('/', 100, true);
148
149     SetProp('|', 1000, true);
150     SetProp('&', 1001, true);
151     SetProp('^', 1002, true);
152
153     SetProp('!', 10000, false);
154     SetProp('~', 10000, false);
155
156     inpt = 0;
157     char c;
158     while((c = getchar()) != EOF && c != '\n' && c != '\r') if(c != '\u') inp[
        inpt++] = c;
159 // 输入.
160 printf("%s\n", inp);
161 // 表达式符号.
162 int len = Destruct(inp, inpt);
163 for(int i=0; i<len; i++) if(sx[i] == 0) printf("."); else printf("%c",
        sx[i]); printf("\n");
164 // 运算数.
165 int t = 0; for(int i=0; i<len; i++) if(sx[i] == 0) printf("%d", aval[t
        ++]); printf("\n");
166 Evaluate(sx, len, aval);
167 // 结果.

```

```

168     printf("%d\n", ap.top());
169
170     return 0;
171 }
172
173 // (123+---213-+---321)+4*-57^6 = -159 correct!

```

并查

```

1  /// 并查集
2
3
4  /// 简易的集合合并并查集,带路径压缩.
5  /// 重新初始化:
6  memset(f, 0, sizeof(int) * (n+1));
7  ///////////////////////////////////////////////////
8  int f[mxn];
9  int findf(int x){ return f[x]==x ? x : f[x]=findf(f[x]); }
10 int connect(int a,int b){ f[findf(a)]=findf(b); }
11
12
13 /// 集合并查集,带路径压缩和按秩合并.
14 /// c[i]: 点i作为集合表头时, 该集合大小.
15 /// 重新初始化:
16 memset(f, 0, sizeof(int) * (n+1));
17 memset(c, 0, sizeof(int) * (n+1));
18 ///////////////////////////////////////////////////
19 int f[mxn];
20 int c[mxn];
21 int connect(int a,int b)
22 {
23     if(c[findf(a)]>c[findf(b)]) // 把b接到a中.
24     { c[findf(a)]+=c[findf(b)]; f[findf(b)] = findf(a); } // 执行顺序不可对
        调.
25     else // 把a接到b中.
26     { c[findf(b)]+=c[findf(a)]; f[findf(a)] = findf(b); }
27 }
28
29
30 /// 集合并查集,带路径压缩,非递归.
31 /// 重新初始化:
32 memset(f, 0, sizeof(int) * (n+1));
33 ///////////////////////////////////////////////////
34 int f[mxn];
35 int findf(int x) // 传入参数x不可为引用.
36 {
37     stack<int> q;
38     while(f[x]!=x) q.push(x), x=f[x];
39     while(!q.empty()) f[q.top()]=x, q.pop();

```

```

40 }
41 void connect(int a,int b){ f[findf(a)]=findf(b); } // *可以换成按秩合并版本
    *

```

可持久化线段树

```

1  /// 可持久化线段树.
2
3  /// 动态开点的权值线段树；查询区间k大；
4  /// 线段树节点记录区间内打上了标记的节点有多少个；只支持插入；不带懒标记.
5  /// 如果要打tag和推tag，参考普通线段树. 注意这样做以后基本就不能支持两棵树相
    减.
6
7  /// 池子大小.
8  const int pg = 4000000;
9
10 /// 树根数量.
11 const int mxn = 105000;
12
13 /// 权值的最大值. 默认线段树的插入范围是 [0, INF].
14 const int INF=(1<<30)-1;
15
16 /// 重新初始化:
17 nt = 0;
18 ...
19 SegmentTreeInit(n);
20
21 //////////////////////////////////////
22
23 struct node
24 {
25     int t;
26     node*l,*r;
27     node(){ t=0; l=r=NULL; }
28     void update() { t=l->t+r->t; }
29 }pool[pg];
30
31 int nt;
32
33 node* newnode() { return &pool[nt++]; }
34
35 node* nil;
36 node* root[mxn];
37
38 void SegmentTreeInit(int size = 0)
39 {
40     nil = newnode();
41     nil->l = nil->r = nil;
42     nil->t = 0;

```

```

43     for(int i=0; i<=size; i++) root[i] = nil;
44 }
45
46 /// 在(子)树y的基础上新建(子)树x, 修改树中位置为cp的值.
47 int cp;
48 node*Change(node*x, node*y, int l = 0, int r = INF)
49 {
50     if(cp<l || r<cp) return y;
51     x=newnode();
52     if(l==r) { x->t = l + y->t; return x; }
53     int mid = (l+r)>>1;
54     x->l = Change(x->l, y->l, l, mid);
55     x->r = Change(x->r, y->r, mid+1, r);
56     x->update();
57     return x;
58 }
59
60 /// 查询树r减去树l的线段树中的第k大.
61 int Query(int ql,int qr,int k)
62 {
63     node*x=root[ql],*y=root[qr];
64     int l=0, r=INF;
65     while(l != r)
66     {
67         int mid = (l+r)>>1;
68         if(k <= x->l->t - y->l->t)
69             r = mid, x = x->l, y = y->l;
70         else
71         {
72             k -= x->l->t - y->l->t;
73             l = mid+1, x = x->r, y = y->r;
74         }
75     }
76     return l;
77 }
78
79 int n;
80
81 int main()
82 {
83
84     int q;
85     scanf("%d",&n);
86     scanf("%d",&q);
87
88     SegmentTreeInit(n);
89
90
91     for(int i=0;i<n;i++)
92     {
93         int c;

```

```

94         scanf("%d",&c);
95         cp=c;
96         root[i+1]=Change(root[i+1],root[i],0,INF);
97     }
98
99
100    for(int i=0;i<q;i++)
101    {
102        int a,b,k;
103        scanf("%d%d%d",&a,&b,&k);
104        printf("%d\n",Query(b,a-1,k));
105    }
106
107    return 0;
108 }

```

轻重边剖分

```

1  /// 轻重边剖分+dfs序.
2  const int mxn = 105000; // 最大节点数.
3
4  /// n: 实际点数.
5  /// c[i]: 顶点i属于的链的编号.
6  /// f[i]: 顶点i的父节点.
7  /// mxi[i]: 记录点i的重边应该连向哪个子节点. 用于dfs序构建.
8  /// sz[i]: 子树i的节点个数.
9  int n;
10 int c[mxn];
11 int f[mxn];
12 int mxi[mxn];
13 int sz[mxn];
14 /// ct: 链数.
15 /// ch[i]: 链头节点编号.
16 int ct;
17 int ch[mxn];
18 /// loc[i]: 节点i在dfs序中的位置.
19 /// til[i]: 子树i在dfs序中的末尾位置.
20 int loc[mxn];
21 int til[mxn];
22
23 /// 操作子树i的信息 <=> 操作线段树上闭区间 loc[i], til[i].
24 /// 操作路径信息 <=> 按照LCA访问方式访问线段树上的点.
25
26 /// 重新初始化:
27 et = pool;
28 for(int i=0; i<n; i++) eds[i] = NULL;
29
30 //////////////////////////////////////
31

```

```

32
33 struct edge{ int in; edge*nxt; } pool[mxn<<1];
34 edge*eds[mxn]; edge*et=pool;
35 void addedge(int a,int b){ et->in=b; et->nxt=eds[a]; eds[a]=et++; }
36 #define FOREACH_EDGE(e,x) for(edge*e=eds[x];e;e=e->nxt)
37 #define FOREACH_SON(e,x) for(edge*e=eds[x];e;e=e->nxt) if(f[x]!=e->in)
38
39 int q[mxn]; int qh,qt;
40 void BuildChain(int root) /// 拓扑序搜索(逆向广搜). 防爆栈.
41 {
42     f[root]=-1; /// 不要修改! 用于在走链时判断是否走到头了.
43     q[qt++]=root;
44     while(qh!=qt) { int x = q[qh++]; FOREACH_SON(e,x) { f[e->in] = x; q[qt
        ++] = e->in;} }
45     for(int i=n-1; i>=0; i--)
46     {
47         int x = q[i];
48         sz[x] = 0;
49         if(!eds[x]) { sz[x] = 1; ch[ct] = x; c[x] = ct++; continue; }
50         int mxp = eds[x]->in;
51         FOREACH_SON(e,x)
52         {
53             sz[x] += sz[e->in];
54             if(sz[e->in] > sz[mxp]) mxp = e->in;
55         }
56         c[x] = c[mxi[x] = mxp]; ch[c[x]] = x;
57     }
58 }
59
60 /// 如果不需要dfs序, 只需要节点所在链的信息, 该函数可以放空.
61 int curl;
62 void BuildDFSOrder(int x)
63 {
64     loc[x] = curl++;
65     if(eds[x]) BuildDFSOrder(mxi[x]); /// dfs序按照重边优先顺序构造, 可以保证
        所有重边在dfs序上连续.
66     FOREACH_SON(e,x) if(e->in != mxi[x]) BuildDFSOrder(e->in);
67     til[x] = curl-1;
68 }
69
70 void HLD(int root)
71 {
72     ct = 0;
73     BuildChain(root);
74     curl = 0;
75     BuildDFSOrder(root);
76 }
77
78 /// 线段树.
79 #define L (x<<1)
80 #define R (x<<1|1)

```

```

81 int t[mxn<<3];
82 int tag[mxn<<3];
83
84 inline void pushtag(int x,int l,int r)
85 {
86     if(tag[x]==0) return;
87     tag[L] = tag[R] = tag[x];
88     int mid = (l+r)>>1;
89     if(tag[x]==-1) { t[L]=t[R]=0; }
90     else if(tag[x]==1) { t[L]=mid-l+1; t[R]=r-mid; }
91     tag[x]=0;
92 }
93 inline void Update(int x,int l,int r)
94 { t[x] = t[L] + t[R]; }
95
96 int cl, cr, cv;
97 void Change(int x=1, int l=0, int r=n-1)
98 {
99     if(cr<l || r<cl) return;
100    if(cl<=l && r<=cr)
101        { tag[x] = cv; t[x] = ( tag[x]==-1 ? 0 : r-l+1 ); return; }
102    pushtag(x,l,r);
103    int mid = (l+r)>>1;
104    Change(L,l,mid); Change(R,mid+1,r); Update(x,l,r);
105 }
106 void Modify(int l,int r,int v) { cl=l; cr=r; cv=v; Change(); }
107
108 int ql,qr;
109 int Query(int x=1, int l=0, int r=n-1)
110 {
111     pushtag(x,l,r);
112     if(qr<l || r<ql) return 0;
113     if(cl<=l && r<=cr) return t[x];
114     int mid = (l+r)>>1;
115     return Query(L,l,mid) + Query(R,mid+1,r);
116 }
117 int GetTotalSum() { return t[1]; }
118
119 /// 修改到根的路径上的信息. 按需更改.
120 void Install(int p)
121 {
122     do{
123         Modify(loc[ch[c[p]]], loc[p], 1);
124         p=f[ch[c[p]]];
125     }
126     while(p!=-1);
127 }
128
129 /// 修改子树信息. 按需更改.
130 void Remove(int p)
131 {

```

```

132     Modify(loc[p], til[p], -1);
133 }

```

手写 bitset

```

1  /*
2     预处理p[i] = 2^i
3     保留N位
4     get(d)获取d位
5     set(d,x)将d位设为x
6     count() 返回1的个数
7     zero() 返回是不是0
8     print() 输出
9  */
10 #define lsix(x) ((x)<<6)
11 #define rsix(x) ((x)>>6)
12 #define msix(x) ((x)-(((x)>>6)<<6))
13 ull p[64] = {1};
14 struct BitSet{
15     ull s[rsix(N-1)+1];
16     int cnt;
17     void resize(int n){
18         if(n>N)n=N;
19         int t = rsix(n-1)+1;
20         if(cnt<t)
21             memset(s+cnt,0,sizeof(ull)*(t-cnt));
22         cnt = t;
23     }
24     BitSet(int n){
25         SET(s,0);
26         cnt=1;
27         resize(n);
28     }
29     BitSet(){cnt=1;SET(s,0);}
30     BitSet operator & (BitSet &that){
31         int len = min(that.cnt, this->cnt);
32         BitSet ans(lsix(len));
33         Repr(i,len)ans.s[i] = this->s[i] & that.s[i];
34         ans.maintain();
35         return ans;
36     }
37     BitSet operator | (BitSet &that){
38         int len = max(that.cnt, this->cnt);
39         BitSet ans(lsix(len));
40         Repr(i,len)ans.s[i] = this->s[i] | that.s[i];
41         ans.maintain();
42         return ans;
43     }
44     BitSet operator ^ (BitSet &that){

```



```

45     int len = max(that.cnt, this->cnt);
46     BitSet ans(lsix(len));
47     Repr(i, len) ans.s[i] = this->s[i] ^ that.s[i];
48     ans.maintain();
49     return ans;
50 }
51 BitSet operator << (int x){
52     int c = rsix(x), r = msix(x);
53     BitSet ans(lsix(cnt+c+(r!=0)));
54     for (int i = min(ans.cnt-1, cnt+c); i-c >= 0; --i){
55         if(i-c < cnt)
56             ans.s[i] = s[i-c] << r;
57         if (r && i-c-1 >= 0) ans.s[i] |= s[i-c-1] >> (64-r);
58     }
59     ans.maintain();
60     return ans;
61 }
62 BitSet operator >> (int x){
63     int c = rsix(x), r = msix(x);
64     BitSet ans(lsix(cnt));
65     if(c >= cnt) return ans;
66     Rep(i, cnt-c){
67         ans.s[i] = s[i+c] >> r;
68         if (r && i+c+1 < cnt) ans.s[i] |= s[i+c+1] << (64-r);
69     }
70     ans.maintain();
71     return ans;
72 }
73 int get(int d){
74     int c = rsix(d), r = msix(d);
75     if(c >= cnt) return 0;
76     return (s[c] & p[r]);
77 }
78 void set(int d, int x){
79     if(d > N) return;
80     int c = rsix(d), r = msix(d);
81     if(c >= cnt)
82         resize(lsix(c+1));
83     if(x && (s[c] & p[r])) return;
84     if(!x && !(s[c] & p[r])) return;
85     s[c] ^= p[r];
86 }
87 int count(){
88     int res=0;
89     Rep(i, cnt){
90         ull x = s[i];
91         while(x){
92             res++;
93             x &= x-1;
94         }
95     }

```

```

96         return res;
97     }
98     void maintain() {
99         while(s[cnt-1]==0&&cnt>1)
100             cnt--;
101         if(lsix(cnt)>N){
102             while(lsix(cnt)>N)cnt--;
103             if(lsix(cnt)<N){
104                 cnt++;
105                 for(int i = 63;i>N-lsix(cnt-1)-1;--i)
106                     if(p[i]&s[cnt-1])s[cnt-1]^=p[i];
107             }
108         }
109     }
110     bool zero() {
111         Rep(i,cnt) if(s[i]) return 0;
112         return 1;
113     }
114     void print() {
115         if(lsix(cnt)<=N){
116             rep(i,N-lsix(cnt)) putchar('0');
117             Repr(j,64) putchar(p[j] & s[cnt-1]?'1':'0');
118         } else {
119             Repr(i,N-lsix(cnt-1)-1)
120                 putchar(p[i] & s[cnt-1]?'1':'0');
121         }
122         Repr(i,cnt-2){
123             ull x = s[i];
124             Repr(j,64) putchar(p[j] & x?'1':'0');
125         }
126         putchar('\n');
127     }
128 };

```

树状数组

```

1  inline int lowbit(int x){return x&-x;}
2  //前缀和,可改前缀最值
3  void update(int d, int x=1){
4      if(!d)return;
5      while(d<=n){
6          T[d]+=x;
7          d+=lowbit(d);
8      }
9  }
10 int ask(int d){
11     int res(0);
12     while(d>0){
13         res+=T[d];

```

```

14         d=lowbit(d);
15     }
16     return res;
17 }

```

线段树

```

1  /// 线段树.
2  /// 带乘法和加法标记.
3  /// 只作为样例解释.
4
5  /// mxn: 区间节点数. 线段树点数是它的四倍.
6  const int mxn = 105000;
7  /// n: 实际节点数.
8  /// a: 初始化列表.
9
10 /// 重新初始化:
11 build(); // 可以不使用初始化数组A.
12
13 //////////////////////////////////////
14
15 ll a[mxn];
16 int n,m;
17 ll MOD;
18
19 #define L (x<<1)
20 #define R (x<<1|1)
21 ll t[mxn<<2]; // 当前真实值.
22 ll tagm[mxn<<2]; // 乘法标记.
23 ll taga[mxn<<2]; // 加法标记. 在乘法之后应用.
24 void pushtag(int x,int l,int r)
25 {
26     if(tagm[x]==1 && taga[x]==0) return;
27     ll &m = tagm[x]; ll &a = taga[x];
28     // 向下合并标记.
29     (tagm[L] *= m) %= MOD;
30     (tagm[R] *= m) %= MOD;
31     taga[L] = (taga[L] * m % MOD + a) % MOD;
32     taga[R] = (taga[R] * m % MOD + a) % MOD;
33     // 修改子节点真实值.
34     int mid = (l+r)>>1;
35     t[L] = (t[L] * m % MOD + (mid-l+1) * a) % MOD;
36     t[R] = (t[R] * m % MOD + (r-mid) * a) % MOD;
37     // 清理当前标记.
38     tagm[x] = 1;
39     taga[x] = 0;
40 }
41
42 /// 从子节点更新当前节点真实值.

```

```

43  /// 以下程序可以保证在Update之前该节点已经没有标记.
44  void update(int x) { t[x] = (t[L] + t[R]) % MOD; }
45
46  void build(int x=1,int l=1,int r=n) // 初始化.
47  {
48      taga[x] = 0; tagm[x] = 1;
49      if(l==r) { t[x] = a[l] % MOD; return; }
50      int mid=(l+r)>>1;
51      build(L,l,mid); build(R,mid+1,r);
52      update(x);
53  }
54
55  int cl,cr; ll cv; int ct;
56  void Change(int x=1,int l=1,int r=n)
57  {
58      if(cr<l || r<cl) return;
59      if(cl<=l && r<=cr) // 是最终访问节点, 修改真实值并打上标记.
60      {
61          if(ct == 1)
62          {
63              (tagm[x] *= cv) %= MOD;
64              (taga[x] *= cv) %= MOD;
65              (t[x] *= cv) %= MOD;
66          }
67          else if(ct == 2)
68          {
69              (taga[x] += cv) %= MOD;
70              (t[x] += (r-l+1) * cv) %= MOD;
71          }
72          return;
73      }
74      pushtag(x,l,r); // 注意不要更改推标记操作的位置.
75      int mid = (l+r)>>1;
76      Change(L,l,mid); Change(R,mid+1,r); update(x);
77  }
78
79  void Modify(int l,int r,ll v,int type)
80  { cl=l; cr=r; cv=v; ct=type; Change(); }
81
82  int ql,qr;
83  ll Query(int x=1,int l=1,int r=n)
84  {
85      if(qr<l || r<ql) return 0;
86      if(ql<=l && r<=qr) return t[x];
87      pushtag(x,l,r); // 注意不要更改推标记操作的位置.
88      int mid=(l+r)>>1;
89      return (Query(L,l,mid) + Query(R,mid+1,r)) % MOD;
90  }
91  ll Getsum(int l,int r)
92  { ql=l; qr=r; return Query(); }
93

```

```

94 void Output(int x=1,int l=1,int r=n,int depth=0)
95 {
96     printf("[%d] [%d,%d] t:%lld m:%lld a:%lld\n",x,l,r,t[x],taga[x],tagm[x]);
97     ;
98     if(l==r) return;
99     int mid=(l+r)>>1;
100     Output(L,l,mid); Output(R,mid+1,r);
101 }
102 int main()
103 {
104     n=getint(); MOD=getint();
105     for(int i=1;i<=n;i++) a[i]=getint();
106     build();
107     m=getint();
108     for(int i=0;i<m;i++)
109     {
110         int type = getint();
111         if(type==3)
112         {
113             int l = getint();
114             int r = getint();
115             printf("%lld\n",Getsum(l,r));
116         }
117         else
118         {
119             int l = getint();
120             int r = getint();
121             int v = getint();
122             Modify(l,r,v,type);
123         }
124     }
125     return 0;
126 }

```

左偏树

```

1 int n,m,root,add;
2 struct node{
3     int key,l,r,fa,add;
4 }heap1[maxn*2+1],heap2[maxn*2+1];
5 void down(int x){
6     heap1[heap1[x].l].key+=heap1[x].add;
7     heap1[heap1[x].l].add+=heap1[x].add;
8     heap1[heap1[x].r].key+=heap1[x].add;
9     heap1[heap1[x].r].add+=heap1[x].add;
10    heap1[x].add=0;
11 }
12 int fa(int x){

```

```

13     int tmp=x;
14     while (heap1[tmp].fa) tmp=heap1[tmp].fa;
15     return tmp;
16 }
17 int sum(int x){
18     int tmp=x,sum=0;
19     while (tmp=heap1[tmp].fa) sum+=heap1[tmp].add;
20     return sum;
21 }
22 int merge1(int x,int y){
23     if (!x || !y) return x?x:y;
24     if (heap1[x].key<heap1[y].key) swap(x,y);
25     down(x);
26     heap1[x].r=merge1(heap1[x].r,y);
27     heap1[heap1[x].r].fa=x;
28     swap(heap1[x].l,heap1[x].r);
29     return x;
30 }
31 int merge2(int x,int y){
32     if (!x || !y) return x?x:y;
33     if (heap2[x].key<heap2[y].key) swap(x,y);
34     heap2[x].r=merge2(heap2[x].r,y);
35     heap2[heap2[x].r].fa=x;
36     swap(heap2[x].l,heap2[x].r);
37     return x;
38 }
39 int del1(int x){
40     down(x);
41     int y=merge1(heap1[x].l,heap1[x].r);
42     if (x==heap1[heap1[x].fa].l) heap1[heap1[x].fa].l=y;else heap1[heap1[x].
        fa].r=y;
43     heap1[y].fa=heap1[x].fa;
44     return fa(y);
45 }
46 void del2(int x){
47     int y=merge2(heap2[x].l,heap2[x].r);
48     if (root==x) root=y;
49     if (x==heap2[heap2[x].fa].l) heap2[heap2[x].fa].l=y;else heap2[heap2[x].
        fa].r=y;
50     heap2[y].fa=heap2[x].fa;
51 }
52 void renew1(int x,int v){
53     heap1[x].key=v;
54     heap1[x].fa=heap1[x].l=heap1[x].r=0;
55 }
56 void renew2(int x,int v){
57     heap2[x].key=v;
58     heap2[x].fa=heap2[x].l=heap2[x].r=0;
59 }
60 //建树
61 int heapify(){

```

```

62     queue<int> Q;
63     for (int i=1;i<=n;++i) Q.push(i);
64     while (Q.size()>1){
65         int x=Q.front();Q.pop();
66         int y=Q.front();Q.pop();
67         Q.push(merge2(x,y));
68     }
69     return Q.front();
70 }
71 //合并两棵树
72 void U(){
73     int x,y;scanf("%d%d",&x,&y);
74     int fx=fa(x),fy=fa(y);
75     if (fx!=fy) if (merge1(fx,fy)==fx) del2(fy);else del2(fx);
76 }
77 //单点修改
78 void A1(){
79     int x,v;scanf("%d%d",&x,&v);
80     del2(fa(x));
81     int y=del1(x);
82     renew1(x,heap1[x].key+v+sum(x));
83     int z=merge1(y,x);
84     renew2(z,heap1[z].key);
85     root=merge2(root,z);
86 }
87 //联通块修改
88 void A2(){
89     int x,v,y;scanf("%d%d",&x,&v);
90     del2(y=fa(x));
91     heap1[y].key+=v;
92     heap1[y].add+=v;
93     renew2(y,heap1[y].key);
94     root=merge2(root,y);
95 }
96 //全局修改
97 void A3(){
98     int v;scanf("%d",&v);
99     add+=v;
100 }
101 //单点查询
102 void F1(){
103     int x;scanf("%d",&x);
104     printf("%d\n",heap1[x].key+sum(x)+add);
105 }
106 //联通块最大值
107 void F2(){
108     int x;scanf("%d",&x);
109     printf("%d\n",heap1[fa(x)].key+add);
110 }
111 //全局最大值
112 void F3(){

```

```

113     printf("%d\n", heap2[root].key+add);
114 }
115 int main(){
116     scanf("%d",&n);
117     for (int i=1;i<=n;++i)
118         scanf("%d",&heap1[i].key), heap2[i].key=heap1[i].key;
119     root=heapify();
120     scanf("%d",&m);
121     for (int i=1;i<=m;++i){
122         scanf("%s",s);
123         if (s[0]=='U') U();
124         if (s[0]=='A'){
125             if (s[1]=='1') A1();
126             if (s[1]=='2') A2();
127             if (s[1]=='3') A3();
128         }
129         if (s[0]=='F'){
130             if (s[1]=='1') F1();
131             if (s[1]=='2') F2();
132             if (s[1]=='3') F3();
133         }
134     }
135     return 0;
136 }

```

动态规划

插头 DP

```

1 //POJ 2411
2 //一个row*col的矩阵，希望用2*1或者1*2的矩形来填满，求填充的总方案数
3 //输入为长和宽
4 #include <cstdio>
5 #include <cstring>
6 #include <algorithm>
7
8 using namespace std;
9 #define LL long long
10
11 const int maxn=2053;
12 struct Node
13 {
14     int H[maxn];
15     int S[maxn];
16     LL N[maxn];
17     int size;
18     void init()
19     {

```



```

20         size=0;
21         memset(H,-1,sizeof(H));
22     }
23     void push(int SS,LL num)
24     {
25         int s=SS%amaxn;
26         while( ~H[s] && S[H[s]]!=SS )
27             s=(s+1)%amaxn;
28
29         if(~H[s])
30         {
31             N[H[s]]+=num;
32         }
33         else
34         {
35             S[size]=SS;
36             N[size]=num;
37             H[s]=size++;
38         }
39     }
40     LL get(int SS)
41     {
42         int s=SS%amaxn;
43         while( ~H[s] && S[H[s]]!=SS )
44             s=(s+1)%amaxn;
45
46         if(~H[s])
47         {
48             return N[H[s]];
49         }
50         else
51         {
52             return 0;
53         }
54     }
55 } dp[2];
56 int now,pre;
57 int get(int S,int p,int l=1)
58 {
59     if(p<0) return 0;
60     return (S>>(p*1))&((1<<l)-1);
61 }
62 void set(int &S,int p,int v,int l=1)
63 {
64     S^=get(S,p,l)<<(p*1);
65     S^=(v&((1<<l)-1))<<(p*1);
66 }
67 int main()
68 {
69     int n,m;
70     while( scanf("%d%d",&n,&m),n||m )

```

```

71     {
72         if (n%2 && m%2) {puts("0");continue;}
73         int now=1,pre=0;
74         dp[now].init();
75         dp[now].push(0,1);
76         for (int i=0;i<n;i++) for (int j=0;j<n;j++)
77         {
78             swap(now,pre);
79             dp[now].init();
80             for (int s=0;s<dp[pre].size;s++)
81             {
82                 int S=dp[pre].S[s];
83                 LL num=dp[pre].N[s];
84                 int p=get(S,j);
85                 int q=get(S,j-1);
86                 int nS=S;
87                 set(nS,j,1-p);
88                 dp[now].push(nS,num);
89                 if (p==0 && q==1)
90                 {
91                     set(S,j-1,0);
92                     dp[now].push(S,num);
93                 }
94             }
95         }
96         printf("%lld\n",dp[now].get(0));
97     }
98 }

```

概率 DP

```

1  /*
2  POJ 2096
3
4  一个软件有s个子系统，会产生n种bug
5  某人一天发现一个bug，这个bug属于一个子系统，属于一个分类
6  每个bug属于某个子系统的概率是1/s，属于某种分类的概率是1/n
7  问发现n种bug，每个子系统都发现bug的天数的期望。
8
9  dp[i][j]表示已经找到i种bug，j个系统的bug，达到目标状态的天数的期望
10 dp[n][s]=0;要求的答案是dp[0][0];
11 dp[i][j]可以转化成以下四种状态：
12     dp[i][j],发现一个bug属于已有的i个分类和j个系统。概率为 (i/n)*(j/s);
13     dp[i][j+1],发现一个bug属于已有的分类，不属于已有的系统。概率为 (i/n)*(1-
        j/s);
14     dp[i+1][j],发现一个bug属于已有的系统，不属于已有的分类，概率为 (1-i/n)*(
        j/s);
15     dp[i+1][j+1],发现一个bug不属于已有的系统，不属于已有的分类，概率为 (1-i/
        n)*(1-j/s);

```

```

16 整理便得到转移方程
17  */
18
19 #include<stdio.h>
20 #include<iostream>
21 #include<algorithm>
22 #include<string.h>
23 using namespace std;
24 const int MAXN = 1010;
25 double dp[MAXN][MAXN];
26
27 int main()
28 {
29     int n, s;
30     while (scanf("%d%d", &n, &s) != EOF)
31     {
32         dp[n][s] = 0;
33         for (int i = n; i >= 0; i--)
34             for (int j = s; j >= 0; j--)
35             {
36                 if (i == n && j == s) continue;
37                 dp[i][j] = (i * (s - j) * dp[i][j + 1] + (n - i) * j * dp[i
+ 1][j] + (n - i) * (s - j) * dp[i + 1][j + 1] + n * s)
/ (n * s - i * j);
38             }
39         printf("%.4lf\n", dp[0][0]);
40     }
41     return 0;
42 }

```

数位 DP

```

1  //HDU-2089 输出不包含4和62的数字的个数
2
3  int dp[10][10];
4  int k = 0;
5  int dig[100];
6
7  void init()
8  {
9      dp[0][0] = 1;
10     for (int i = 1; i <= 7; i++){
11         for (int j = 0; j < 10; j++){
12             for (int k = 0; k < 10; k++){
13                 if (j != 4 && !(j == 6 && k == 2)){
14                     dp[i][j] += dp[i - 1][k];
15                 }
16             }
17         }

```

```

18     }
19 }
20
21 int solve (int num)
22 {
23     int ret = num, ans = 0;
24     memset(dig, 0, sizeof(dig));
25     k = 1;
26     while (ret > 0)
27     {
28         dig[k++] = ret % 10;
29         ret /= 10;
30     }
31     for (int i = k; i > 0; i--)
32     {
33         for (int j = 0; j < dig[i]; j++)
34         {
35             if (!(j == 2 && dig[i + 1] == 6) && j != 4)
36             {
37                 ans += dp[i][j];
38             }
39         }
40         if (dig[i] == 4 || (dig[i] == 2 && dig[i + 1] == 6))
41         {
42             break;
43         }
44     }
45     return ans;
46 }
47
48 int main() {
49     int n, m;
50     init();
51     while (cin >> n >> m && (n + m))
52     {
53         int ans = solve(m + 1) - solve(n);
54         cout << ans << endl;
55     }
56     return 0;
57 }

```

四边形 DP

```

1  /*HDOJ2829
2  题目大意：给定一个长度为n的序列，至多将序列分成m段，每段序列都有权值，权值为
   序列内两个数两两相乘之和。m≤n≤1000. 令权值最小。
3  状态转移方程：
4  dp[c][i]=min(dp[c][i],dp[c-1][j]+w[j+1][i])
5  url->:http://blog.csdn.net/bnmjnz/article/details/41308919

```

```

6  */
7
8  #include <iostream>
9  #include <cstdio>
10 #include <cstring>
11 using namespace std;
12 const int INF = 1 << 30;
13 const int MAXN = 1000 + 10;
14 typedef long long LL;
15 LL dp[MAXN][MAXN]; //dp[c][j]表示前j个点切了c次后的最小权值
16 int val[MAXN];
17 int w[MAXN][MAXN]; //w[i][j]表示i到j无切割的权值
18 int s[MAXN][MAXN]; //s[c][j]表示前j个点切的第c次的位置
19 int sum[MAXN];
20 int main()
21 {
22     int n, m;
23     while (~scanf("%d%d", &n, &m))
24     {
25         if (n == 0 && m == 0) break;
26         memset(s, 0, sizeof(s));
27         memset(w, 0, sizeof(w));
28         memset(dp, 0, sizeof(dp));
29         memset(sum, 0, sizeof(sum));
30         for (int i = 1; i <= n; ++i)
31         {
32             scanf("%d", &val[i]);
33             sum[i] += sum[i - 1] + val[i];
34         }
35         for (int i = 1; i <= n; ++i)
36         {
37             w[i][i] = 0;
38             for (int j = i + 1; j <= n; ++j)
39             {
40                 w[i][j] = w[i][j - 1] + val[j] * (sum[j - 1] - sum[i - 1]);
41             }
42         }
43         for (int i = 1; i <= n; ++i)
44         {
45             for (int j = 1; j <= m; ++j)
46             {
47                 dp[j][i] = INF;
48             }
49         }
50         for (int i = 1; i <= n; ++i)
51         {
52             dp[0][i] = w[1][i];
53             s[0][i] = 0;
54         }
55         for (int c = 1; c <= m; ++c)
56         {

```

```

57         s[c][n + 1] = n; //设置边界
58         for (int i = n; i > c; --i)
59         {
60             int tmp = INF, k;
61             for (int j = s[c - 1][i]; j <= s[c][i + 1]; ++j)
62             {
63                 if (dp[c - 1][j] + w[j + 1][i] < tmp)
64                 {
65                     tmp = dp[c - 1][j] + w[j + 1][i]; //状态转移方程, j
66                     //之前切了c-1次, 第c次切j到j+1间的
67                     k = j;
68                 }
69             }
70             dp[c][i] = tmp;
71             s[c][i] = k;
72         }
73         printf("%d\n", dp[m][n]);
74     }
75     return 0;
76 }

```

斜率 DP

```

1 //HDU 3507
2 //给出n,m, 求在n个数中分成任意段, 每段的花销是(sigma(a[l],a[r])+m)^2, 求最小
  值
3 //http://acm.hdu.edu.cn/showproblem.php?pid=3507
4
5 #include <stdio.h>
6 #include <iostream>
7 #include <string.h>
8 #include <queue>
9 using namespace std;
10 const int MAXN = 500010;
11
12 int dp[MAXN];
13 int q[MAXN];
14 int sum[MAXN];
15
16 int head, tail, n, m;
17
18 int getDP(int i, int j)
19 {
20     return dp[j] + m + (sum[i] - sum[j]) * (sum[i] - sum[j]);
21 }
22
23 int getUP(int j, int k)
24 {

```

```

25     return dp[j] + sum[j] * sum[j] - (dp[k] + sum[k] * sum[k]);
26 }
27 int getDOWN(int j, int k)
28 {
29     return 2 * (sum[j] - sum[k]);
30 }
31
32 int main()
33 {
34     while (scanf("%d%d", &n, &m) == 2)
35     {
36         for (int i = 1; i <= n; i++)
37             scanf("%d", &sum[i]);
38         sum[0] = dp[0] = 0;
39         for (int i = 1; i <= n; i++)
40             sum[i] += sum[i - 1];
41         head = tail = 0;
42         q[tail++] = 0;
43         for (int i = 1; i <= n; i++)
44         {
45             while (head + 1 < tail && getUP(q[head + 1], q[head]) <= sum[i]
46                 *getDOWN(q[head + 1], q[head]))
47                 head++;
48             dp[i] = getDP(i, q[head]);
49             while (head + 1 < tail && getUP(i, q[tail - 1]) *getDOWN(q[tail - 1], q[tail - 2]) <= getUP(q[tail - 1], q[tail - 2]) *
50                 getDOWN(i, q[tail - 1]))
51                 tail--;
52             q[tail++] = i;
53         }
54         printf("%d\n", dp[n]);
55     }
56 }

```

状压 DP

```

1 //CF 580D
2 //有n种菜，选m种。每道菜有一个权值，有些两个菜按顺序挨在一起会有combo的权值
3 //加成。求最大权值
4 #include <bits/stdc++.h>
5 using namespace std;
6 const int maxn = 20;
7 typedef long long LL;
8
9 int a[maxn];
10 int comb[maxn][maxn];
11 LL dp[(1 << 18) + 10][maxn];

```

```

12 LL ans = 0;
13 int n, m, k;
14
15 int Cnt(int st)
16 {
17     int res = 0;
18     for (int i = 0; i < n; i++)
19     {
20         if (st & (1 << i))
21         {
22             res++;
23         }
24     }
25     return res;
26 }
27
28 int main()
29 {
30     memset(comb, 0, sizeof comb);
31     scanf("%d%d%d", &n, &m, &k);
32     for (int i = 0; i < n; i++)
33     {
34         scanf("%d", &a[i]);
35     }
36     for (int i = 0; i < k; i++)
37     {
38         int x, y, c;
39         scanf("%d%d%d", &x, &y, &c);
40         x--;
41         y--;
42         comb[x][y] = c;
43     }
44     int end = (1 << n);
45     memset(dp, 0, sizeof dp);
46     for (int st = 0; st < end; st++)
47     {
48         for (int i = 0; i < n; i++)
49         {
50             if (st & (1 << i))
51             {
52                 bool has = false;
53                 for (int j = 0; j < n; j++)
54                 {
55                     if (j != i && (st & (1 << j)))
56                     {
57                         has = true;
58                         dp[st][i] = max(dp[st][i], dp[st ^ (1 << i)][j] + a[
59                             i] + comb[j][i]);
60                     }
61                 }
62                 if (!has)

```



```

62         {
63             dp[st][i] = a[i];
64         }
65     }
66     if (Cnt(st) == m)
67     {
68         ans = max(ans, dp[st][i]);
69     }
70 }
71 }
72
73 cout << ans << endl;
74 return 0;
75 }

```

最长上升子序列

```

1  //使用lisDP查找,a为待查找串,b用于返回结果串,n为a的长度
2  int dpSearch(int num, int low, int high)
3  {
4      int mid;
5      while (low <= high)
6      {
7          mid = (low + high) / 2;
8          if (num >= b[mid]) low = mid + 1;
9          else high = mid - 1;
10     }
11     return low;
12 }
13
14 int lisDP(int* a, int* b, int n)
15 {
16     int i, len, pos;
17     b[1] = a[1];
18     len = 1;
19     for (i = 2; i <= n; i++)
20     {
21         if (a[i] >= b[len])
22         {
23             len = len + 1;
24             b[len] = a[i];
25         }
26         else
27         {
28             pos = dpSearch(a[i], 1, len);
29             b[pos] = a[i];
30         }
31     }
32     return len;

```

33 }

图论

best's therom

```
1  /*
2     以某个点为起点的欧拉回路数=该点为根的树形图数*(所有点出度-1)的乘积
3     从1出发的欧拉回路数量
4     重边当作多种方案
5  */
6  #include <algorithm>
7  #include <cmath>
8  #include <cstdio>
9  #include <cstring>
10 #include <iostream>
11 #include <map>
12 #include <queue>
13 #include <set>
14 #include <stack>
15 #include <string>
16 #include <vector>
17
18 #define each(i, n) for (int(i) = 0; (i) < (n); (i)++)
19 #define reach(i, n) for (int(i) = n - 1; (i) >= 0; (i)--)
20 #define range(i, st, en) for (int(i) = (st); (i) <= (en); (i)++)
21 #define rrange(i, st, en) for (int(i) = (en); (i) >= (st); (i)--)
22 #define fill(ary, num) memset((ary), (num), sizeof(ary))
23
24 using namespace std;
25 typedef long long ll;
26
27 const int maxn = 410;
28 const int mod = 998244353;
29
30 int d[maxn][maxn], g[maxn][maxn];
31 ll c[maxn][maxn];
32 int in[maxn], mul[(int)2e5 + 10], out[maxn];
33
34 int n;
35
36 ll getDet(ll a[][maxn], int n)
37 {
38     range(i, 1, n) range(j, 1, n) a[i][j] = (a[i][j] + mod) % mod;
39     ll ret = 1;
40     range(i, 2, n)
41     {
42         range(j, i + 1, n) while (a[j][i])
```

```

43     {
44         ll t = a[i][i] / a[j][i];
45         range(k, i, n) a[i][k] = (a[i][k] - a[j][k] * t % mod + mod) %
            mod;
46         range(k, i, n) swap(a[i][k], a[j][k]);
47         ret = -ret;
48     }
49     if (a[i][i] == 0)
50         return 0;
51     ret = ret * a[i][i] % mod;
52 }
53 return (ret + mod) % mod;
54 }
55
56 ll fastPow(ll n, ll m)
57 {
58     ll ans = 1;
59     while (m) {
60         if (m & 1)
61             ans = ans * n % mod;
62         n = n * n % mod;
63         m >>= 1;
64     }
65     return ans;
66 }
67
68 bool judgeEuler()
69 {
70     range(i, 1, n) if (in[i] != out[i]) return false;
71     return true;
72 }
73
74 int main()
75 {
76     int cas = 0;
77     mul[0] = mul[1] = 1;
78     range(i, 2, (int)(2e5 + 5)) mul[i] = (mul[i - 1] * 1LL * i) % mod;
79     while (scanf("%d", &n) != EOF) {
80         fill(in, 0), fill(d, 0), fill(out, 0);
81         range(i, 1, n) range(j, 1, n)
82         {
83             scanf("%d", &g[i][j]);
84             d[j][j] += g[i][j];
85             in[j] += g[i][j];
86             out[i] += g[i][j];
87         }
88         if (!judgeEuler()) {
89             printf("Case_#%d: 0\n", ++cas);
90             continue;
91         } else if (n == 1) {
92             printf("Case_#%d: %d\n", ++cas, mul[g[1][1]]);

```

```

93         continue;
94     }
95     range(i, 1, n) range(j, 1, n) c[i][j] = d[i][j] - g[i][j];
96     ll trees = getDet(c, n) % mod * mul[in[1]] % mod;
97     range(i, 2, n) trees = trees * mul[in[i] - 1] % mod;
98     range(i, 1, n) range(j, 1, n) trees = trees * fastPow(mul[g[i][j]],
99                 mod - 2) % mod;
100     printf("Case_#%d: %lld\n", ++cas, trees);
101 }
102 return 0;
103 }
104 /*
105 欧拉回路：每条边恰走一次的回路
106 欧拉通路：每条边恰走一次的路径
107 欧拉图：存在欧拉回路的图
108 半欧拉图：存在欧拉通路的图
109 有向欧拉图：每个点入度=出度
110 无向欧拉图：每个点度数为偶数
111 有向半欧拉图：一个点入度=出度+1，一个点入度=出度-1，其他点入度=出度
112 无向半欧拉图：两个点度数为奇数，其他点度数为偶数
113 */

```

k 短路可持久化堆

```

1  /*
2   s到t的k短路
3  */
4  typedef long long LL ;
5  typedef pair < int , int > pii ;
6  typedef pair < LL , int > pli ;
7  typedef unsigned long long ULL ;
8
9  #define clr( a , x ) memset ( a , x , sizeof a )
10 #define st first
11 #define ed second
12
13 const int MAXN = 10005 ;
14 const int BLOCK = 22 ;
15 const LL INF = 1e18 ;
16
17 namespace Leftist_Tree {
18     struct Node {
19         int l , r , x , h ;
20         LL val ;
21     } T[MAXN * 200] ;
22     int Root[MAXN] ;
23     int node_num ;
24     int newnode ( const Node& o ) {
25         T[node_num] = o ;

```

```

26         return node_num ++ ;
27     }
28     void init () {
29         node_num = 1 ;
30         T[0].l = T[0].r = T[0].x = T[0].h = 0 ;
31         T[0].val = INF ;
32     }
33     int merge ( int x , int y ) {
34         if ( !x ) return y ;
35         if ( T[x].val > T[y].val ) swap ( x , y ) ;
36         int o = newnode ( T[x] ) ;
37         T[o].r = merge ( T[o].r , y ) ;
38         if ( T[T[o].l].h < T[T[o].r].h ) swap ( T[o].l , T[o].r ) ;
39         T[o].h = T[T[o].r].h + 1 ;
40         return o ;
41     }
42     void insert ( int& x , LL val , int v ) {
43         int o = newnode ( T[0] ) ;
44         T[o].val = val , T[o].x = v ;
45         x = merge ( x , o ) ;
46     }
47     void show ( int o ) {
48         printf ( "%d_%ld_%ld_%ld\n" , o , T[o].val , T[T[o].l].val , T[T[
49             o].r].val ) ;
50         if ( T[o].l ) show ( T[o].l ) ;
51         if ( T[o].r ) show ( T[o].r ) ;
52     }
53 }
54 using namespace Leftist_Tree ;
55 vector < pii > G[MAXN] , E[MAXN] ;
56 int vis[MAXN] ;
57 int in[MAXN] , p[MAXN] ;
58 LL d[MAXN] ;
59 int s , t ;
60 int n , m , k ;
61
62 void addedge ( int u , int v , int c ) {
63     G[u].push_back ( pii ( v , c ) ) ;
64     E[v].push_back ( pii ( u , c ) ) ;
65 }
66
67 void dij () {
68     priority_queue < pli > q ;
69     d[t] = 0 ;
70     q.push ( pli ( 0 , t ) ) ;
71     while ( !q.empty () ) {
72         int u = q.top ().ed ;
73         q.pop () ;
74         if ( vis[u] ) continue ;
75         vis[u] = 1 ;

```

```

76         for ( int i = 0 ; i < E[u].size () ; ++ i ) {
77             int v = E[u][ i ].st ;
78             if ( d[v] > d[u] + E[u][ i ].ed ) {
79                 p[v] = u ;
80                 d[v] = d[u] + E[u][ i ].ed ;
81                 q.push ( pli ( -d[v] , v ) ) ;
82             }
83         }
84     }
85 }
86
87 void dfs ( int u ) {
88     if ( vis[u] ) return ;
89     vis[u] = 1 ;
90     if ( p[u] ) Root[u] = Root[p[u]] ;
91     int flag = 1 ;
92     for ( int i = 0 ; i < G[u].size () ; ++ i ) {
93         int v = G[u][ i ].st ;
94         if ( d[v] == INF ) continue ;
95         if ( p[u] == v && d[u] == G[u][ i ].ed + d[v] && flag ) {
96             flag = 0 ;
97             continue ;
98         }
99         LL val = d[v] - d[u] + G[u][ i ].ed ;
100         insert ( Root[u] , val , v ) ;
101     }
102     for ( int i = 0 ; i < E[u].size () ; ++ i ) {
103         if ( p[E[u][ i ].st] == u ) dfs ( E[u][ i ].st ) ;
104     }
105 }
106
107 void solve () {
108     for ( int i = 1 ; i <= n ; ++ i ) {
109         G[i].clear () ;
110         E[i].clear () ;
111         d[i] = INF ;
112         vis[i] = 0 ;
113         p[i] = 0 ;
114     }
115     for ( int i = 0 ; i < m ; ++ i ) {
116         int u , v , c ;
117         scanf ( "%d%d%d" , &u , &v , &c ) ;
118         addedge ( u , v , c ) ;
119     }
120     scanf ( "%d%d%d" , &s , &t , &k ) ;
121     dij () ;
122     if ( d[s] == INF ) {
123         printf ( "-1\n" ) ;
124         return ;
125     }
126     if ( s != t ) — k ;

```

```

127     if ( !k ) {
128         printf ( "%lld\n" , d[s] ) ;
129         return ;
130     }
131     for ( int i = 1 ; i <= n ; ++ i ) {
132         vis[i] = 0 ;
133     }
134     init () ;
135     Root[t] = 0 ;
136     dfs ( t ) ;
137     priority_queue < pli , vector < pli > , greater < pli > > q ;
138     if ( Root[s] ) q.push ( pli ( d[s] + T[Root[s]].val , Root[s] ) ) ;
139     while ( k — ) {
140         if ( q.empty () ) {
141             printf ( "-1\n" ) ;
142             return ;
143         }
144         pli u = q.top () ;
145         q.pop () ;
146         if ( !k ) {
147             printf ( "%lld\n" , u.st ) ;
148             return ;
149         }
150         int x = T[u.ed].l , y = T[u.ed].r , v = T[u.ed].x ;
151         if ( Root[v] ) q.push ( pli ( u.st + T[Root[v]].val , Root[v] ) ) ;
152         if ( x ) q.push ( pli ( u.st + T[x].val - T[u.ed].val , x ) ) ;
153         if ( y ) q.push ( pli ( u.st + T[y].val - T[u.ed].val , y ) ) ;
154     }
155 }
156
157 int main () {
158     while ( ~scanf ( "%d%d" , &n , &m ) ) solve () ;
159     return 0 ;
160 }

```

spfa 费用流

```

1  /*
2      调用minCostMaxflow(s,t,cost)返回s到t的最大流,cost保存费用
3      多组数据调用Ginit()
4  */
5  struct E{
6      int v,n,F,f,cost;
7  }G[M];
8  int point[N],cnt;
9  int pre[N];
10 int dis[N];
11 bool vis[N];
12 void Ginit(){

```

```

13     cnt=1;
14     SET(point,0);
15 }
16 void addedge(int u,int v,int F,int cost){
17     G[++cnt]=(E){v,point[u],F,0,cost},point[u]=cnt;
18     G[++cnt]=(E){u,point[v],0,0,-cost},point[v]=cnt;
19 }
20 bool spfa(int s,int t){
21     queue<int>q;
22     SET(vis,0);
23     SET(pre,0);
24     repab(i,s,t)
25         dis[i]=infi;
26     dis[s]=0;
27     vis[s]=1;
28     q.push(s);
29     while(!q.empty()){
30         int u=q.front();q.pop();
31         vis[u]=0;
32         for(int i=point[u];i;i=G[i].n){
33             int v=G[i].v;
34             if(G[i].F>G[i].f&&dis[v]-dis[u]-G[i].cost>0){
35                 dis[v]=dis[u]+G[i].cost;
36                 pre[v]=i;
37                 if(!vis[v]){
38                     vis[v]=1;
39                     q.push(v);
40                 }
41             }
42         }
43     }
44     return pre[t];
45 }
46 int minCostMaxflow(int s,int t,int &cost){
47     int f=0;
48     cost=0;
49     while(spfa(s,t)){
50         int Min=infi;
51         for(int i=pre[t];i;i=pre[G[i^1].v]){
52             if(Min>G[i].F-G[i].f)
53                 Min=G[i].F-G[i].f;
54         }
55         for(int i=pre[t];i;i=pre[G[i^1].v]){
56             G[i].f+=Min;
57             G[i^1].f-=Min;
58             cost+=G[i].cost*Min;
59         }
60         f+=Min;
61     }
62     return f;
63 }

```


Tarjan 有向图强连通分量

```
1  /*
2     调用SCC()得到强连通分量,调用suodian()缩点
3     belong[i]为所在scc编号,sccnum为scc数量
4     原图用addege,存在G,缩点后的图用addege2,存在G1
5     多组数据时调用Ginit()
6  */
7  int n, m;
8  int point[N], cnt;
9  int low[N], dfn[N], belong[N], Stack[N];
10 bool instack[N];
11 int dfsnow, Stop, sccnum;
12 struct E{
13     int u, v, nex;
14 }G[M], G1[M];
15 void tarjan(int u){
16     int v;
17     dfn[u] = low[u] = ++dfsnow;
18     instack[u] = 1;
19     Stack[++Stop] = u;
20     for (int i = point[u]; i; i = G[i].nex){
21         v = G[i].v;
22         if (!dfn[v]){
23             tarjan(v);
24             low[u] = min(low[u], low[v]);
25         }
26         else
27             if (instack[v])
28                 low[u] = min(low[u], dfn[v]);
29     }
30     if (dfn[u] == low[u]){
31         sccnum++;
32         do{
33             v = Stack[Stop--];
34             instack[v] = 0;
35             belong[v] = sccnum;
36             num[sccnum][++num[sccnum][0]] = v;
37         }
38         while (v != u);
39     }
40 }
41 void Ginit(){
42     cnt = 0;
43     SET(point, 0);
44 }
45 void SCC(){
```

```

46     Stop = scnum = dfsnow = 0;
47     SET(dfn, 0);
48     rep(i,n)
49         if (!dfn[i])
50             tarjan(i);
51 }
52 void addedge(int a, int b){
53     G[++cnt] = (E){a,b,point[a]}, point[a] = cnt;
54 }
55 void addedge2(int a, int b){
56     G1[++cnt] = (E){a,b,point[a]}, point[a] = cnt;
57 }
58 int degree[N];
59 void suodian(){
60     Ginit();
61     SET(degree,0);
62     rep(i,m)
63         if (belong[G[i].u] != belong[G[i].v]){
64             addedge2(belong[G[i].u], belong[G[i].v]);
65             degree[belong[G[i].v]]++;
66         }
67 }
68 /*
69     割点和桥
70     割点：删除后使图不连通
71     桥(割边)：删除后使图不连通
72     对图深度优先搜索，定义DFS(u)为u在搜索树(以下简称为树)中被遍历到的次序
        号。定义Low(u)为u或u的子树中能通过非树边追溯到的DFS序号最小的节点。
73     ( )= { ( )}; ( ),( , )为非树边; ( ),( , )为树边}
74     一个顶点u是割点，当且仅当满足(1)或(2)
75     (1) u为树根，且u有多于一个子树。(2) u不为树根，且满足存在(u,v)为树边，
        使得DFS(u)<=Low(v)。
76     一条无向边(u,v)是桥，当且仅当(u,v)为树边，且满足DFS(u)<Low(v)。
77 */

```

zkw 费用流

```

1  /*
2      调用zkw(s,t,cost)返回s到t的最大流,cost保存费用
3      多组数据调用Ginit()
4  */
5  struct E{
6      int v,n,F,f,c;
7  }G[M];
8  int point[N],cnt;
9  int dis[N];
10 bool vis[N];
11 void Ginit(){
12     cnt=1;

```

```

13     SET(point,0);
14 }
15 void addedge(int u,int v,int F,int cost){
16     G[++cnt]=(E){v,point[u],F,0,cost},point[u]=cnt;
17     G[++cnt]=(E){u,point[v],0,0,-cost},point[v]=cnt;
18 }
19 bool spfa(int s,int t){
20     queue<int>q;
21     SET(vis,0);
22     repab(i,s,t)
23         dis[i]=infi;
24     dis[s]=0;
25     vis[s]=1;
26     q.push(s);
27     while(!q.empty()){
28         int u=q.front();q.pop();
29         vis[u]=0;
30         for(int i=point[u];i;i=G[i].n){
31             int v=G[i].v;
32             if(G[i].F>G[i].f&&dis[v]-dis[u]-G[i].c>0){
33                 dis[v]=dis[u]+G[i].c;
34                 if(!vis[v]){
35                     vis[v]=1;
36                     q.push(v);
37                 }
38             }
39         }
40     }
41     return dis[t]!=infi;
42 }
43 bool mark[N];
44 int dfs(int u,int t,int f,int &ans){
45     mark[u]=1;
46     if(u==t)return f;
47     double w;
48     int used=0;
49     for(int i=point[u];i;i=G[i].n){
50         if(G[i].F>G[i].f&&!mark[G[i].v]&&dis[u]+G[i].c-dis[G[i].v]==0){
51             w=dfs(G[i].v,t,min(G[i].F-G[i].f,f-used),ans);
52             G[i].f+=w;
53             G[i^1].f-=w;
54             ans+=G[i].c*w;
55             used+=w;
56             if(used==f)return f;
57         }
58     }
59     return used;
60 }
61 int zkw(int s,int t,int &ans){
62     int tmp=0;
63     ans=0;

```

```

64     while(spfa(s,t)){
65         mark[t]=1;
66         while(mark[t]){
67             SET(mark,0);
68             tmp+=dfs(s,t,infi,ans);
69         }
70     }
71     return tmp;
72 }

```

倍增 LCA

```

1  /*
2     调用init(),且处理出dep数组后
3     调用lca(x,y)得到x,y的lca
4  */
5  int p[M], f[N][M];
6  void init(){
7      p[0] = 1;
8      rep(i,M-1){
9          p[i] = p[i-1]<<1;
10         rep(j,n)
11             if(f[j][i-1])
12                 f[j][i] = f[f[j][i-1]][i-1]
13     }
14 }
15 int lca(int x,int y){
16     if(dep[x] > dep[y])
17         swap(x, y);
18     if(dep[x] < dep[y])
19         Rep(i,M)
20             if((dep[y] - dep[x]) & p[i])
21                 y = f[y][i];
22     Repr(i,M)
23         if(f[x][i] != f[y][i]){
24             x = f[x][i];
25             y = f[y][i];
26         }
27     if(x != y)
28         return f[x][0];
29     return x;
30 }

```

点分治

```

1  /*
2     问有多少对点它们两者间的距离小于等于K

```

```

3  */
4  #include <algorithm>
5  #include <cstring>
6  #include <cstdio>
7  #include <bitset>
8  #include <queue>
9  using namespace std;
10 #define N 40002
11 int n, K, dis[N], point[N], cnt, siz[N], maxs[N], r, son[N], ans;
12 bitset<N> vis;
13 struct E
14 {
15     int v, w, next;
16 }G[N<1];
17 inline void add(int u, int v, int w)
18 {
19     G[++cnt] = (E){v, w, point[u]}, point[u] = cnt;
20     G[++cnt] = (E){u, w, point[v]}, point[v] = cnt;
21 }
22 inline void getroot(int u, int f)
23 {
24     siz[u] = 1, maxs[u] = 0;
25     for (int i = point[u]; i; i = G[i].next)
26     {
27         if (G[i].v == f || vis[G[i].v]) continue;
28         getroot(G[i].v, u);
29         siz[u] += siz[G[i].v];
30         maxs[u] = max(maxs[u], siz[G[i].v]);
31     }
32     maxs[u] = max(maxs[u], n-siz[u]);
33     if (maxs[r] > maxs[u])
34         r = u;
35 }
36 queue<int> Q;
37 bitset<N> hh;
38 inline void bfs(int u)
39 {
40     hh.reset();
41     Q.push(u);
42     hh[u] = 1;
43     while (!Q.empty())
44     {
45         int i = Q.front(); Q.pop();
46         for (int p = point[i]; p; p = G[p].next)
47         {
48             if (hh[G[p].v] || vis[G[p].v]) continue;
49             son[++son[0]] = dis[G[p].v] = dis[i] + G[p].w;
50             hh[G[p].v] = 1;
51             Q.push(G[p].v);
52         }
53     }

```

```

54 }
55 /*inline void dfs(int u, int f)
56 {
57     for (int i = point[u]; i; i = G[i].next)
58     {
59         if (G[i].v == f || vis[G[i].v]) continue;
60         son[++son[0]] = dis[G[i].v] = dis[u] + G[i].w;
61         dfs(G[i].v, u);
62     }
63 }*/
64 inline int calc(int u)
65 {
66     int res(0), i;
67     son[son[0]=1] = dis[u], bfs(u);
68     sort(son+1, son+son[0]+1);
69     son[++son[0]] = 1<<30;
70     for (i = 1; i <= son[0]; ++i)
71     {
72         if (son[i] > K) continue;
73         int x = upper_bound(son+1, son+1+son[0], K-son[i])-(son);
74         res += x-1;
75         if (son[i] << 1 <= K) res--;
76     }
77     return res;
78 }
79 inline void solve(int u)
80 {
81     dis[u] = 0, vis[u] = 1;
82     ans += calc(u);
83     for (int i = point[u]; i; i = G[i].next)
84     {
85         if (vis[G[i].v]) continue;
86         dis[G[i].v] = G[i].w, ans -= calc(G[i].v);
87         n = siz[G[i].v];
88         maxs[r=0] = N, getroot(G[i].v, 0);
89         solve(r);
90     }
91 }
92 int main()
93 {
94     int i, j, u, v, w;
95     scanf("%d", &n);
96     memset(point, 0, sizeof(point));
97     vis.reset();
98     for (i = 1; i < n; ++i)
99     {
100         scanf("%d_%d_%d", &u, &v, &w);
101         add(u, v, w);
102     }
103     scanf("%d", &K);
104     maxs[r=0]=n+1;

```

```

105     getroot(1, 0);
106     solve(r);
107     printf("%d\n", ans>>1);
108     ans = 0;
109     return 0;
110 }
111 /*
112     给一棵树,每条边有权.求一条简单路径,权值和等于K,且边的数量最小
113 */
114 #include <cstdio>
115 #include <cstring>
116 #include <bitset>
117 #include <algorithm>
118 using namespace std;
119 #define N 200005
120 #define Max (N<<1)
121 bitset<Max> vis;
122 struct hh
123 {
124     int i, x;
125     bool operator < (const hh &nb) const
126     {
127         return x < nb.x;
128     }
129 }son[N];
130 int n, K, siz[N], maxs[N], dfn[N], point[N], belong[N], dis[N], dep[N], cnt,
    r, ans(Max);
131 char c;
132 inline void read(int &x)
133 {
134     for (c = getchar(); c > '9' || c < '0'; c = getchar());
135     for (x = 0; c >= '0' && c <= '9'; c = getchar())
136         x = (x << 3) + (x << 1) + c - '0';
137 }
138 struct E
139 {
140     int v, w, next;
141 }G[N<<1];
142 inline void add(int u, int v, int w)
143 {
144     G[++cnt] = (E){v, w, point[u]}, point[u] = cnt;
145     G[++cnt] = (E){u, w, point[v]}, point[v] = cnt;
146 }
147 inline void getroot(int u, int f)
148 {
149     siz[u] = 1, maxs[u] = 0;
150     for (int i = point[u]; i; i = G[i].next)
151     {
152         int v = G[i].v;
153         if (v == f || vis[v])continue;
154         getroot(v, u);

```

```

155         siz[u] += siz[v], maxs[u] = max(maxs[u], siz[v]);
156     }
157     maxs[u] = max(maxs[u], n-siz[u]);
158     if (maxs[u] < maxs[r]) r = u;
159 }
160 inline void dfs(int u, int f)
161 {
162     if (f != r) belong[u] = belong[f];
163     for (int i = point[u]; i; i = G[i].next)
164     {
165         int v = G[i].v;
166         if (v == f || vis[v]) continue;
167         dep[v] = dep[u] + 1;
168         son[++son[0].i].x = dis[v] = dis[u] + G[i].w;
169         son[son[0].i].i = v;
170         dfs(v, u);
171     }
172     dfn[u] = ++cnt;
173 }
174 inline int calc(int u)
175 {
176     int res(Max);
177     son[++son[0].i].x = dis[u];
178     son[1].i = u;
179     belong[u] = u;
180     for (int i = point[u]; i; i = G[i].next)
181     {
182         int v = G[i].v;
183         if (vis[v]) continue;
184         belong[v] = v;
185     }
186     dfs(u, 0);
187     sort(son+1, son+1+son[0].i);
188     son[++son[0].i].x = K < 1;
189     for (int i = 1; i <= son[0].i; ++i)
190     {
191         son[i].x = K - son[i].x;
192         int x = lower_bound(son+1, son+1+son[0].i, son[i])-(son);
193         for (; son[i].x == son[x].x; ++x)
194         {
195             if (x == i) continue;
196             if (belong[son[i].i] == belong[son[x].i]) continue;
197             res = min(res, dep[son[i].i]-dep[u]+dep[son[x].i]-dep[u]);
198         }
199         son[i].x = K - son[i].x;
200     }
201     return res;
202 }
203 inline void solve(int u)
204 {
205     son[0].i = dis[u] = 0;

```



```

206     vis[u] = 1;
207     ans = min(ans, calc(u));
208     for (int i = point[u]; i; i = G[i].next)
209     {
210         int v = G[i].v;
211         if (vis[v]) continue;
212         maxs[r=0] = N-1;
213         n = siz[v];
214         getroot(v, 0);
215         solve(r);
216     }
217 }
218 int main()
219 {
220     // freopen("a.in", "r", stdin);
221     int i, u, v, w;
222     read(n), read(K);
223     // scanf("%d %d", &n, &K);
224     for (i = 1; i < n; ++i)
225     {
226         read(u), read(v), read(w);
227         //scanf("%d %d %d", &u, &v, &w);
228         add(u+1, v+1, w);
229     }
230     maxs[cnt=r=0] = N-1;
231     getroot(1, 0);
232     solve(r);
233     printf("%d\n", ans == Max ? -1 : ans);
234 }

```

堆优化 dijkstra

```

1  /*
2     调用Dijkstra(s)得到从s出发的最短路,存在dist中
3     多组数据时调用Ginit()
4  */
5  struct qnode{
6      int v,c;
7      bool operator <(const qnode &r) const{
8          return c>r.c;
9      }
10 };
11 struct E{
12     int v,w,n;
13 }G[M];
14 int point[N], cnt;
15 bool vis[N];
16 int dist[N];
17 void Dijkstra(int s){

```

```

18     SET(vis,0);
19     SET(dist,127);
20     dist[s]=0;
21     priority_queue<qnode> que;
22     while(!que.empty()) que.pop();
23     que.push((qnode){s,0});
24     qnode tmp;
25     while(!que.empty()){
26         tmp=que.top();
27         que.pop();
28         int u=tmp.v;
29         if(vis[u]) continue;
30         vis[u]=1;
31         for_each_edge(u){
32             int v = G[i].v;
33             if(!vis[v]&&dist[v]>dist[u]+G[i].w){
34                 dist[v]=dist[u]+G[i].w;
35                 que.push((qnode){v,dist[v]});
36             }
37         }
38     }
39 }
40 void addedge(int u,int v,int w){
41     G[++cnt] = (E){v,w,point[u]}, point[u] = cnt;
42 }
43 void Ginit(){
44     cnt = 0;
45     SET(point,0);
46 }

```

矩阵树定理

```

1  /*
2      矩阵树定理
3      令g为度数矩阵,a为邻接矩阵
4      生成树的个数为g-a的任何一个n-1阶主子式的行列式的绝对值
5      det(a,n)返回n阶矩阵a的行列式
6      所以直接调用det(g-a,n-1)就得到答案
7      O(n^3)
8      有取模版和double版
9      无向图生成树的个数与根无关
10     有必选边时压缩边
11     有向图以i为根的树形图的数目=基尔霍夫矩阵去掉第i行和第i列的主子式的行列式
        的值(即Matrix-Tree定理不仅适用于求无向图生成树数目,也适用于求有向图
        树形图数目)
12 */
13 int det(int a[N][N], int n){
14     rep(i,n)
15         rep(j,n)

```

```

16         a[i][j]=(a[i][j]+mod)%mod;
17     ll ans=1,f=1;
18     rep(i,n){
19         repab(j,i+1,n){
20             ll A=a[i][i],B=a[j][i];
21             while(B!=0){
22                 ll t=A/B,A%=B;swap(A,B);
23                 repab(k,i,n)
24                     a[i][k]=(a[i][k]-t*a[j][k]%mod+mod)%mod;
25                 repab(k,i,n)
26                     swap(a[i][k],a[j][k]);
27                 f=-f;
28             }
29         }
30         if(!a[i][i])return 0;
31         ans=ans*a[i][i]%mod;
32     }
33     if(f==-1)return (mod-ans)%mod;
34     return ans;
35 }
36 double det(double a[N][N],int n){
37     int i,j,k,sign=0;
38     double ret=1,t;
39     for(i=1;i<=n;i++){
40         for(j=1;j<=n;j++){
41             b[i][j]=a[i][j];
42         }
43         for(i=1;i<=n;i++){
44             if(zero(b[i][i])){
45                 for(j=i+1;j<=n;j++){
46                     if(!zero(b[j][i]))
47                         break;
48                 }
49                 if(j>n)
50                     return 0;
51                 for(k=i;k<=n;k++){
52                     t=b[i][k],b[i][k]=b[j][k],b[j][k]=t;
53                     sign++;
54                 }
55                 ret*=b[i][i];
56                 for(k=i+1;k<=n;k++){
57                     b[i][k]/=b[i][i];
58                 }
59                 for(j=i+1;j<=n;j++){
60                     for(k=i+1;k<=n;k++){
61                         b[j][k]-=b[j][i]*b[i][k];
62                     }
63                 }
64                 if(sign&1)
65                     ret=-ret;
66                 return ret;
67             }
68         }
69     }
70     /*
71     最小生成树计数
72     */

```

```

67 #define dinf 1e10
68 #define linf (LL)1<<60
69 #define LL long long
70 #define clr(a,b) memset(a,b,sizeof(a))
71 LL mod;
72 struct Edge{
73     int a,b,c;
74     bool operator<(const Edge & t)const{
75         return c<t.c;
76     }
77 }edge[M];
78 int n,m;
79 LL ans;
80 int fa[N],ka[N],vis[N];
81 LL gk[N][N],tmp[N][N];
82 vector<int>gra[N];
83 int findfa(int a,int b[]){return a==b[a]?a:b[a]=findfa(b[a],b);}
84 LL det(LL a[][N],int n){
85     for(int i=0;i<n;i++)for(int j=0;j<n;j++)a[i][j]%mod;
86     long long ret=1;
87     for(int i=1;i<n;i++){
88         for(int j=i+1;j<n;j++){
89             while(a[j][i]){
90                 LL t=a[i][i]/a[j][i];
91                 for(int k=i;k<n;k++){
92                     a[i][k]=(a[i][k]-a[j][k]*t)%mod;
93                 }
94                 swap(a[i][k],a[j][k]);
95                 ret=-ret;
96             }
97             if(a[i][i]==0)return 0;
98             ret=ret*a[i][i]%mod;
99             //ret%=mod;
100     }
101     return (ret+mod)%mod;
102 }
103 int main(){
104     while(scanf("%d%d%d",&n,&m,&mod)==3){
105         if(n==0 && m==0 && mod==0)break;
106         memset(gk,0,sizeof(gk));
107         memset(tmp,0,sizeof(tmp));
108         memset(fa,0,sizeof(fa));
109         memset(ka,0,sizeof(ka));
110         memset(tmp,0,sizeof(tmp));
111         for(int i=0;i<N;i++)gra[i].clear();
112         for(int i=0;i<m;i++){
113             scanf("%d%d%d",&edge[i].a,&edge[i].b,&edge[i].c);
114             sort(edge,edge+m);
115             for(int i=1;i<=n;i++)fa[i]=i,vis[i]=0;
116             int pre=-1;
117             ans=1;

```

```

118     for (int h=0;h<=m;h++){
119         if (edge[h].c!=pre || h==m) {
120             for (int i=1;i<=n;i++){
121                 if (vis[i]) {
122                     int u=findfa(i,ka);
123                     gra[u].push_back(i);
124                     vis[i]=0;
125                 }
126             for (int i=1;i<=n;i++){
127                 if (gra[i].size()>1){
128                     for (int a=1;a<=n;a++){
129                         for (int b=1;b<=n;b++){
130                             tmp[a][b]=0;
131                         }
132                     int len=gra[i].size();
133                     for (int a=0;a<len;a++){
134                         for (int b=a+1;b<len;b++){
135                             int la=gra[i][a],lb=gra[i][b];
136                             tmp[a][b]=(tmp[b][a]-gk[la][lb]);
137                             tmp[a][a]+=gk[la][lb];tmp[b][b]+=gk[la][lb];
138                         }
139                     long long ret=(long long)det(tmp,len);
140                     ret%=mod;
141                     ans=(ans*ret%mod)%mod;
142                     for (int a=0;a<len;a++)fa[gra[i][a]]=i;
143                 }
144             for (int i=1;i<=n;i++){
145                 ka[i]=fa[i]=findfa(i,fa);
146                 gra[i].clear();
147             }
148             if (h==m) break;
149             pre=edge[h].c;
150         }
151         int a=edge[h].a,b=edge[h].b;
152         int pa=findfa(a,fa),pb=findfa(b,fa);
153         if (pa==pb) continue;
154         vis[pa]=vis[pb]=1;
155         ka[findfa(pa,ka)]=findfa(pb,ka);
156         gk[pa][pb]++;gk[pb][pa]++;
157     }
158     int flag=0;
159     for (int i=2;i<=n&&!flag;i++)if (ka[i]!=ka[i-1]) flag=1;
160     ans%=mod;
161     printf("%I64d\n",flag?0:ans);
162 }
163 return 0;

```

平面欧几里得距离最小生成树

```

1  #include<cstdio>
2  #include<cstdlib>
3  #include<cstring>
4  #include<algorithm>
5  #include<iostream>
6  #include<fstream>
7  #include<map>
8  #include<ctime>
9  #include<list>
10 #include<set>
11 #include<queue>
12 #include<cmath>
13 #include<vector>
14 #include<bitset>
15 #include<functional>
16 #define x first
17 #define y second
18 #define mp make_pair
19 #define pb push_back
20 using namespace std;
21
22 typedef long long LL;
23 typedef double ld;
24
25 const int MAX=400000+10;
26 const int NUM=20;
27
28 int n;
29
30 struct point
31 {
32     LL x,y;
33     int num;
34     point() {}
35     point(LL a,LL b)
36     {
37         x=a;
38         y=b;
39     }
40 }d[MAX];
41
42 int operator < (const point& a,const point& b)
43 {
44     if(a.x!=b.x) return a.x<b.x;
45     else return a.y<b.y;
46 }
47
48 point operator - (const point& a,const point& b)
49 {
50     return point(a.x-b.x,a.y-b.y);
51 }

```

```

52
53 LL chaji(const point& s,const point& a,const point& b)
54 {
55     return (a.x-s.x)*(b.y-s.y)-(a.y-s.y)*(b.x-s.x);
56 }
57
58 LL dist(const point& a,const point& b)
59 {
60     return (a.x-b.x)*(a.x-b.x)+(b.y-a.y)*(b.y-a.y);
61 }
62
63 struct point3
64 {
65     LL x,y,z;
66     point3(){}
67     point3(LL a,LL b,LL c)
68     {
69         x=a;
70         y=b;
71         z=c;
72     }
73     point3(point a)
74     {
75         x=a.x;
76         y=a.y;
77         z=x*x+y*y;
78     }
79 };
80
81 point3 operator - (const point3 a,const point3& b)
82 {
83     return point3(a.x-b.x,a.y-b.y,a.z-b.z);
84 }
85
86 point3 chaji(const point3& a,const point3& b)
87 {
88     return point3(a.y*b.z-a.z*b.y,-a.x*b.z+a.z*b.x,a.x*b.y-a.y*b.x);
89 }
90
91 LL dianji(const point3& a,const point3& b)
92 {
93     return a.x*b.x+a.y*b.y+a.z*b.z;
94 }
95
96 LL in_circle(point a,point b,point c,point d)
97 {
98     if(chaji(a,b,c)<0)
99         swap(b,c);
100     point3 aa(a),bb(b),cc(c),dd(d);
101     bb=bb-aa;cc=cc-aa;dd=dd-aa;
102     point3 f=chaji(bb,cc);

```

```

103     return dianji(dd, f);
104 }
105
106 struct Edge
107 {
108     int t;
109     list<Edge>::iterator c;
110     Edge() {}
111     Edge(int v)
112     {
113         t=v;
114     }
115 };
116 list<Edge> ne[MAX];
117
118 void add(int a,int b)
119 {
120     ne[a].push_front(b);
121     ne[b].push_front(a);
122     ne[a].begin()->c=ne[b].begin();
123     ne[b].begin()->c=ne[a].begin();
124 }
125
126 int sign(LL a)
127 {
128     return a>0?1:(a==0?0:-1);
129 }
130
131 int cross(const point& a,const point& b,const point& c,const point& d)
132 {
133     return sign(chaji(a,c,b))*sign(chaji(a,b,d))>0 && sign(chaji(c,a,d))*
        sign(chaji(c,d,b))>0;
134 }
135
136 void work(int l,int r)
137 {
138     int i,j,nowl=l,nowr=r;
139     list<Edge>::iterator it;
140     if(l+2>=r)
141     {
142         for(i=l;i<=r;++i)
143             for(j=i+1;j<=r;++j)
144                 add(i,j);
145         return;
146     }
147     int mid=(l+r)/2;
148     work(l,mid);work(mid+1,r);
149     int flag=1;
150     for(;flag;)
151     {
152         flag=0;

```



```

153 point ll=d[nowl],rr=d[nowr];
154 for(it=ne[nowl].begin();it!=ne[nowl].end();++it)
155 {
156     point t=d[it->t];
157     LL s=chaji(rr,ll,t);
158     if(s>0 || (s==0 && dist(rr,t)<dist(rr,ll)))
159     {
160         nowl=it->t;
161         flag=1;
162         break;
163     }
164 }
165 if(flag)
166     continue;
167 for(it=ne[nowr].begin();it!=ne[nowr].end();++it)
168 {
169     point t=d[it->t];
170     LL s=chaji(ll,rr,t);
171     if(s<0 || (s==0 && dist(ll,rr)>dist(ll,t)))
172     {
173         nowr=it->t;
174         flag=1;
175         break;
176     }
177 }
178 }
179 add(nowl,nowr);
180 for(;1;)
181 {
182     flag=0;
183     int best=0,dir=0;
184     point ll=d[nowl],rr=d[nowr];
185     for(it=ne[nowl].begin();it!=ne[nowl].end();++it)
186         if(chaji(ll,rr,d[it->t])>0 && (best==0 || in_circle(ll,rr,d[
            best],d[it->t])<0))
187             best=it->t,dir=-1;
188     for(it=ne[nowr].begin();it!=ne[nowr].end();++it)
189         if(chaji(rr,d[it->t],ll)>0 && (best==0 || in_circle(ll,rr,d[
            best],d[it->t])<0))
190             best=it->t,dir=1;
191     if(!best)break;
192     if(dir==1)
193     {
194         for(it=ne[nowl].begin();it!=ne[nowl].end();)
195             if(cross(ll,d[it->t],rr,d[best]))
196             {
197                 list<Edge>::iterator ij=it;
198                 ++ij;
199                 ne[it->t].erase(it->c);
200                 ne[nowl].erase(it);
201                 it=ij;

```

```

202         }
203         else ++it;
204         nowl=best;
205     }
206     else if (dir==1)
207     {
208         for (it=ne[nowr].begin(); it!=ne[nowr].end(); )
209             if (cross(rr,d[it->t],ll,d[best]))
210             {
211                 list<Edge>::iterator ij=it;
212                 ++ij;
213                 ne[it->t].erase(it->c);
214                 ne[nowl].erase(it);
215                 it=ij;
216             }
217         else ++it;
218         nowr=best;
219     }
220     add(nowl,nowr);
221 }
222 }
223
224 struct MstEdge
225 {
226     int x,y;
227     LL w;
228 }e[MAX];
229 int m;
230
231 int operator < (const MstEdge& a,const MstEdge& b)
232 {
233     return a.w<b.w;
234 }
235
236 int fa[MAX];
237
238 int findfather(int a)
239 {
240     return fa[a]==a?a:fa[a]=findfather(fa[a]);
241 }
242
243 int Hash[MAX],p[MAX/4][NUM],deep[MAX],place[MAX];
244 LL dd[MAX/4][NUM];
245
246 vector<int> ne2[MAX];
247 queue<int> q;
248
249 LL getans(int u,int v)
250 {
251     if (deep[u]<deep[v])
252         swap(u,v);

```

```

253 LL ans=0;
254 int s=NUM-1;
255 while (deep[u]>deep[v])
256 {
257     while(s && deep[p[u][s]]<deep[v])—s;
258     ans=max(dd[u][s],ans);
259     u=p[u][s];
260 }
261 s=NUM-1;
262 while(u!=v)
263 {
264     while(s && p[u][s]==p[v][s])—s;
265     ans=max(dd[u][s],ans);
266     ans=max(dd[v][s],ans);
267     u=p[u][s];
268     v=p[v][s];
269 }
270 return ans;
271 }
272
273 int main()
274 {
275 #ifndef ONLINE_JUDGE
276     freopen("input.txt","r",stdin);freopen("output.txt","w",stdout);
277 #endif
278     int i,j,u,v;
279     scanf("%d",&n);
280     for(i=1;i<=n;++i)
281     {
282         cin>>d[i].x>>d[i].y;
283         d[i].num=i;
284     }
285     sort(d+1,d+n+1);
286     for(i=1;i<=n;++i)
287         place[d[i].num]=i;
288     work(1,n);
289     for(i=1;i<=n;++i)
290         for(list<Edge>::iterator it=ne[i].begin();it!=ne[i].end();++it)
291         {
292             if(it->t<i) continue;
293             ++m;
294             e[m].x=i;
295             e[m].y=it->t;
296             e[m].w=dist(d[e[m].x],d[e[m].y]);
297         }
298     sort(e+1,e+m+1);
299     for(i=1;i<=n;++i)
300         fa[i]=i;
301     for(i=1;i<=m;++i)
302         if(findfather(e[i].x)!=findfather(e[i].y))
303         {

```

```

304         fa[findfather(e[i].x)]=findfather(e[i].y);
305         ne2[e[i].x].pb(e[i].y);
306         ne2[e[i].y].pb(e[i].x);
307     }
308     q.push(1);
309     deep[1]=1;
310     Hash[1]=1;
311     while(!q.empty())
312     {
313         u=q.front();q.pop();
314         for(i=0;i<(int)ne2[u].size();++i)
315         {
316             v=ne2[u][i];
317             if(!Hash[v])
318             {
319                 Hash[v]=1;
320                 p[v][0]=u;
321                 dd[v][0]=dist(d[u],d[v]);
322                 deep[v]=deep[u]+1;
323                 q.push(v);
324             }
325         }
326     }
327     for(i=1;(1<i)<=n;++i)
328         for(j=1;j<=n;++j)
329         {
330             p[j][i]=p[p[j][i-1]][i-1];
331             dd[j][i]=max(dd[j][i-1],dd[p[j][i-1]][i-1]);
332         }
333     int m;
334     scanf("%d",&m);
335     while(m--)
336     {
337         scanf("%d%d",&u,&v);
338         printf("%.10lf\n",sqrt((ld)getans(place[u],place[v])));
339     }
340     return 0;
341 }

```

最大流 Dinic

```

1  /*
2      调用maxflow()返回最大流
3      S,T为源汇
4      addedge(u,v,f,F)F为反向流量
5      多组数据时调用Ginit()
6  */
7  struct E{
8      int v, f, F, n;

```

```

9  }G[M];
10 int point[N], D[N], cnt, S, T;
11 void Ginit() {
12     cnt = 1;
13     SET(point, 0);
14 }
15 void addedge(int u, int v, int f, int F) {
16     G[++cnt] = (E){v, 0, f, point[u]}, point[u] = cnt;
17     G[++cnt] = (E){u, 0, F, point[v]}, point[v] = cnt;
18 }
19 queue<int> q;
20 int BFS() {
21     SET(D, 0);
22     q.push(S);
23     D[S] = 1;
24     while (!q.empty()) {
25         int u = q.front(); q.pop();
26         for_each_edge(u)
27             if (G[i].F > G[i].f) {
28                 int v = G[i].v;
29                 if (!D[v]) {
30                     D[v] = D[u] + 1;
31                     q.push(v);
32                 }
33             }
34     }
35     return D[T];
36 }
37 int Dinic(int u, int F) {
38     if (u == T) return F;
39     int f = 0;
40     for_each_edge(u) {
41         if (F <= f) break;
42         int v = G[i].v;
43         if (G[i].F > G[i].f && D[v] == D[u] + 1) {
44             int temp = Dinic(v, min(F - f, G[i].F - G[i].f));
45             if (temp == 0)
46                 D[v] = 0;
47             else {
48                 f += temp;
49                 G[i].f += temp;
50                 G[i^1].f -= temp;
51             }
52         }
53     }
54     return f;
55 }
56 int maxflow() {
57     int f = 0;
58     while (BFS())
59         f += Dinic(S, inf);

```

```

60     return f;
61 }
62 /*
63 最大权闭合子图
64     在一个有向无环图中,每个点都有一个权值。
65     现在需要选择一个子图,满足若一个点被选,其后继所有点也会被选。最大化选出的点权和。
66     建图方法:源向所有正权点连容量为权的边,所有负权点向汇点连容量为权的绝对值的边。若原图中存在有向边 $\langle u, v \rangle$ ,则从u向v连容量为正无穷的边。答案为所有正权点和 - 最大流
67 最大权密度子图
68     在一个带点权带边权无向图中,选出一个子图,使得该子图的点权和与边权和的比值最大。
69     二分答案k,问题转为最大化 $|V| - k|E|$ 
70     确定二元关系:如果一条边连接的两个点都被选择,则将获得该边的权值(可能需要处理负权)
71 二分图最小点权覆盖集
72     点覆盖集:在无向图 $G=(V,E)$ 中,选出一个点集 $V'$ ,使得对于任意 $\langle u, v \rangle$ 属于 $E$ ,都有u属于 $V'$ 或v属于 $V'$ ,则称 $V'$ 是无向图G的一个点覆盖集。
73     最小点覆盖集:在无向图中,包含点数最少的点覆盖集被称为最小点覆盖集。
74     这是一个NPC问题,但在二分图中可以用最大匹配模型快速解决。
75
76     最小点权覆盖集:在最小点覆盖集的基础上每个点均被赋上一个点权。
77     建模方法:对二分图进行黑白染色,源点向白点连容量为该点点权的边,黑点向汇点连容量为该点点权的边,对于无向边 $\langle u, v \rangle$ ,设u为白点,则从u向v连容量为正无穷的边。最小割即为答案。
78 二分图最大点权独立集
79     点独立集:在无向图 $G=(V,E)$ 中,选出一个点集 $V'$ ,使得对于任意 $u, v$ 属于 $V'$ , $\langle u, v \rangle$ 不属于 $E$ ,则称 $V'$ 是无向图G的一个点独立集。
80     最大点独立集:在无向图中,包含点数最多的点独立集被称为最大点独立集。
81      $|最大独立集| = |V| - |最大匹配数|$ 
82     这是一个NPC问题,但在二分图中可以用最大匹配模型快速解决。
83     最大点权独立集:在最大点独立集的基础上每个点均被赋上一个点权。
84     建模方法:对二分图进行黑白染色,源点向白点连容量为该点点权的边,黑点向汇点连容量为该点点权的边,对于无向边 $\langle u, v \rangle$ ,设u为白点,则从u向v连容量为正无穷的边。所有点权-最小割即为答案。
85 最小路径覆盖
86     在一个DAG中,用尽量少的不相交的简单路径覆盖所有的节点。
87     最小路径覆盖数=点数-路径中的边数
88     建立一个二分图,把原图中的所有节点分成两份(X集合为i,Y集合为i'),如果原来图中有 $i \rightarrow j$ 的有向边,则在二分图中建立 $i \rightarrow j'$ 的有向边。最终|最小路径覆盖|=|V|-|最大匹配数|
89
90 无源汇可行流
91     建图方法:
92     首先建立附加源点ss和附加汇点tt,对于原图中的边 $x \rightarrow y$ ,若限制为 $[b, c]$ ,那么连边 $x \rightarrow y$ ,流量为 $c-b$ ,对于原图中的某一个点i,记 $d(i)$ 为流入这个点的所有边的下界和减去流出这个点的所有边的下界和
93     若 $d(i) > 0$ ,那么连边 $ss \rightarrow i$ ,流量为 $d(i)$ ,若 $d(i) < 0$ ,那么连边 $i \rightarrow tt$ ,流量为 $-d(i)$ 
94     求解方法:
95     在新图上跑ss到tt的最大流,若新图满流,那么一定存在一种可行流,此时,原

```

图中每一条边的流量应为新图中对应的边的流量+这条边的流量下界

96 有源汇可行流

97 建图方法:在原图中添加一条边 $t \rightarrow s$, 流量限制为 $[0, \text{inf}]$,即让源点和汇点也满足流量平衡条件,这样就改造成了无源汇的网络流图,其余方法同上

98 求解方法:同 无源汇可行流

99 有源汇最大流

100 建图方法:同有源汇可行流

101 求解方法:在新图上跑 ss 到 tt 的最大流,若新图满流,那么一定存在一种可行流,记此时 $\text{sigma } f(s, i) = \text{sum1}$,将 $t \rightarrow s$ 这条边拆掉,在新图上跑 s 到 t 的最大流,记此时 $\text{sigma } f(s, i) = \text{sum2}$,最终答案即为 $\text{sum1} + \text{sum2}$

102 有源汇最小流

103 建图方法:同 无源汇可行流

104 求解方法:求 $ss \rightarrow tt$ 最大流,连边 $t \rightarrow s$, inf ,求 $ss \rightarrow tt$ 最大流,答案即为边 $t \rightarrow s$, inf 的实际流量

105 有源汇费用流

106 建图方法:首先建立附加源点 ss 和附加汇点 tt ,对于原图中的边 $x \rightarrow y$,若限制为 $[b, c]$,费用为 cost ,那么连边 $x \rightarrow y$,流量为 $c - b$,费用为 cost ,对于原图中的某一个点 i ,记 $d(i)$ 为流入这个点的所有边的下界和减去流出这个点的所有边的下界和,若 $d(i) > 0$,那么连边 $ss \rightarrow i$,流量为 $d(i)$,费用为 0 ,若 $d(i) < 0$,那么连边 $i \rightarrow tt$,流量为 $-d(i)$,费用为 0 ,连边 $t \rightarrow s$,流量为 inf ,费用为 0

107 求解方法:跑 $ss \rightarrow tt$ 的最小费用最大流,答案即为(求出的费用+原图中边的下界*边的费用)

108 注意:有上下界的费用流指的是在满足流量限制条件和流量平衡条件的情况下的最小费用流,而不是在满足流量限制条件和流量平衡条件并且满足最大流的情况下的最小费用流,也就是说,有上下界的费用流只需要满足网络流的条件就可以了,而普通的费用流是满足一般条件并且满足是最大流的基础上的最小费用*/

最大团

```

1  /*
2     用二维bool数组a[][]保存邻接矩阵,下标0~n-1
3     建图:Maxclique G = Maxclique(a, n)
4     求最大团:mcqdyn(保存最大团中点的数组, 保存最大团中点数的变量)
5  */
6  typedef bool BB[N];
7  struct Maxclique {
8      const BB* e; int pk, level; const float Tlimit;
9      struct Vertex{ int i, d; Vertex(int i):i(i),d(0){} };
10     typedef vector<Vertex> Vertices; typedef vector<int> ColorClass;
11     Vertices V; vector<ColorClass> C; ColorClass QMAX, Q;
12     static bool desc_degree(const Vertex &vi, const Vertex &vj){
13         return vi.d > vj.d;
14     }
15     void init_colors(Vertices &v){
16         const int max_degree = v[0].d;
17         for(int i = 0; i < (int)v.size(); i++) v[i].d = min(i, max_degree) +
18             1;
19     }

```

```

19 void set_degrees(Vertices &v){
20     for(int i = 0, j; i < (int)v.size(); i++)
21         for(v[i].d = j = 0; j < (int)v.size(); j++)
22             v[i].d += e[v[i].i][v[j].i];
23 }
24 struct StepCount{ int i1, i2; StepCount():i1(0),i2(0){} };
25 vector<StepCount> S;
26 bool cut1(const int pi, const ColorClass &A){
27     for(int i = 0; i < (int)A.size(); i++) if (e[pi][A[i]]) return true;
28     return false;
29 }
30 void cut2(const Vertices &A, Vertices &B){
31     for(int i = 0; i < (int)A.size() - 1; i++)
32         if(e[A.back().i][A[i].i])
33             B.push_back(A[i].i);
34 }
35 void color_sort(Vertices &R){
36     int j = 0, maxno = 1, min_k = max((int)QMAX.size() - (int)Q.size() +
37         1, 1);
38     C[1].clear(), C[2].clear();
39     for(int i = 0; i < (int)R.size(); i++) {
40         int pi = R[i].i, k = 1;
41         while(cut1(pi, C[k])) k++;
42         if(k > maxno) maxno = k, C[maxno + 1].clear();
43         C[k].push_back(pi);
44         if(k < min_k) R[j++].i = pi;
45     }
46     if(j > 0) R[j - 1].d = 0;
47     for(int k = min_k; k <= maxno; k++)
48         for(int i = 0; i < (int)C[k].size(); i++)
49             R[j].i = C[k][i], R[j++].d = k;
50 }
51 void expand_dyn(Vertices &R){// diff -> diff with no dyn
52     S[level].i1 = S[level].i1 + S[level - 1].i1 - S[level].i2;//diff
53     S[level].i2 = S[level - 1].i1;//diff
54     while((int)R.size()) {
55         if((int)Q.size() + R.back().d > (int)QMAX.size()){
56             Q.push_back(R.back().i); Vertices Rp; cut2(R, Rp);
57             if((int)Rp.size()){
58                 if((float)S[level].i1 / ++pk < Tlimit) degree_sort(Rp);
59                 //diff
60                 color_sort(Rp);
61                 S[level].i1++, level++;//diff
62                 expand_dyn(Rp);
63                 level--;//diff
64             }
65             else if((int)Q.size() > (int)QMAX.size()) QMAX = Q;
66             Q.pop_back();
67         }
68     }
69     else return;
70     R.pop_back();

```



```

68     }
69 }
70 void mcqdyn(int* maxclique, int &sz){
71     set_degrees(V); sort(V.begin(), V.end(), desc_degree); init_colors(V)
72     ;
73     for(int i = 0; i < (int)V.size() + 1; i++) S[i].i1 = S[i].i2 = 0;
74     expand_dyn(V); sz = (int)QMAX.size();
75     for(int i = 0; i < (int)QMAX.size(); i++) maxclique[i] = QMAX[i];
76 }
77 void degree_sort(Vertices &R){
78     set_degrees(R); sort(R.begin(), R.end(), desc_degree);
79 }
80 Maxclique(const BB* conn, const int sz, const float tt = 0.025) \
81 : pk(0), level(1), Tlimit(tt){
82     for(int i = 0; i < sz; i++) V.push_back(Vertex(i));
83     e = conn, C.resize(sz + 1), S.resize(sz + 1);
84 }
};

```

最小度限制生成树

```

1  /*
2     只限制一个点的度数
3  */
4  #include <iostream>
5  #include <cstdio>
6  #include <cmath>
7  #include <vector>
8  #include <cstring>
9  #include <algorithm>
10 #include <string>
11 #include <set>
12 #include <ctime>
13 #include <queue>
14 #include <map>
15
16 #define CL(arr, val)    memset(arr, val, sizeof(arr))
17 #define REP(i, n)       for((i) = 0; (i) < (n); ++(i))
18 #define FOR(i, l, h)    for((i) = (l); (i) <= (h); ++(i))
19 #define FORD(i, h, l)   for((i) = (h); (i) >= (l); --(i))
20 #define L(x)            (x) << 1
21 #define R(x)            (x) << 1 | 1
22 #define MID(l, r)       (l + r) >> 1
23 #define Min(x, y)       x < y ? x : y
24 #define Max(x, y)       x < y ? y : x
25 #define E(x)            (1 << (x))
26
27 const double eps = 1e-8;
28 typedef long long LL;

```

```

29 using namespace std;
30 const int inf = ~0u>>2;
31 const int N = 33;
32
33 int parent[N];
34 int g[N][N];
35 bool flag[N][N];
36 map<string, int> NUM;
37
38 int n, k, cnt, ans;
39
40 struct node {
41     int x;
42     int y;
43     int v;
44 } a[1<<10];
45
46 struct edge {
47     int x;
48     int y;
49     int v;
50 } dp[N];
51
52 bool cmp(node a, node b) {
53     return a.v < b.v;
54 }
55
56 int find(int x) { //并查集查找
57     int k, j, r;
58     r = x;
59     while(r != parent[r]) r = parent[r];
60     k = x;
61     while(k != r) {
62         j = parent[k];
63         parent[k] = r;
64         k = j;
65     }
66     return r;
67 }
68
69 int get_num(string s) { //求编号
70     if(NUM.find(s) == NUM.end()) {
71         NUM[s] = ++cnt;
72     }
73     return NUM[s];
74 }
75
76 void kruskal() { //...
77     int i;
78     FOR(i, 1, n) {
79         if(a[i].x == 1 || a[i].y == 1) continue;

```

```

80         int x = find(a[i].x);
81         int y = find(a[i].y);
82         if(x == y) continue;
83         flag[a[i].x][a[i].y] = flag[a[i].y][a[i].x] = true;
84         parent[y] = x;
85         ans += a[i].v;
86     }
87     //printf("%d\n", ans);
88 }
89
90 void dfs(int x, int pre) {    //dfs求1到某节点路程上的最大值
91     int i;
92     FOR(i, 2, cnt) {
93         if(i != pre && flag[x][i]) {
94             if(dp[i].v == -1) {
95                 if(dp[x].v > g[x][i])    dp[i] = dp[x];
96                 else {
97                     dp[i].v = g[x][i];
98                     dp[i].x = x;    //记录这条边
99                     dp[i].y = i;
100                 }
101             }
102             dfs(i, x);
103         }
104     }
105 }
106
107 void init() {
108     ans = 0; cnt = 1;
109     CL(flag, false);
110     CL(g, -1);
111     NUM["Park"] = 1;
112     for(int i = 0; i < N; ++i)    parent[i] = i;
113 }
114
115 int main() {
116     //freopen("data.in", "r", stdin);
117
118     int i, j, v;
119     string s;
120     scanf("%d", &n);
121     init();
122     for(i = 1; i <= n; ++i) {
123         cin >> s;
124         a[i].x = get_num(s);
125         cin >> s;
126         a[i].y = get_num(s);
127         scanf("%d", &v);
128         a[i].v = v;
129         if(g[a[i].x][a[i].y] == -1)    g[a[i].x][a[i].y] = g[a[i].y][a[i].x]
            ] = v;

```

```

130         else    g[a[i].x][a[i].y] = g[a[i].y][a[i].x] = min(g[a[i].x][a[i].y
131             ], v);
132     }
133     scanf("%d", &k);
134     int set[N], Min[N];
135     REP(i, N)    Min[i] = inf;
136     sort(a + 1, a + n + 1, cmp);
137     kruskal();
138     FOR(i, 2, cnt) {    //找到1到其他连通块的最小值
139         if(g[1][i] != -1) {
140             int x = find(i);
141             if(Min[x] > g[1][i]) {
142                 Min[x] = g[1][i];
143                 set[x] = i;
144             }
145         }
146     }
147     int m = 0;
148     FOR(i, 1, cnt) {    //把1跟这些连通块连接起来
149         if(Min[i] != inf) {
150             m++;
151             flag[1][set[i]] = flag[set[i]][1] = true;
152             ans += g[1][set[i]];
153         }
154     }
155     //printf("%d\n", ans);
156     for(i = m + 1; i <= k; ++i) {    //从度为m+1一直枚举到最大为k, 找ans的最小
157         值
158         CL(dp, -1);
159         dp[1].v = -inf;    //dp初始化
160         for(j = 2; j <= cnt; ++j) {
161             if(flag[1][j])    dp[j].v = -inf;
162         }
163         dfs(1, -1);
164         int tmp, mi = inf;
165         for(j = 2; j <= cnt; ++j) {
166             if(g[1][j] != -1) {
167                 if(mi > g[1][j] - dp[j].v) {    //找到一条dp到连通块中某个点的
168                     边, 替换原来连通块中的边 ( 前提是新找的这条边比原来连
169                     通块中那条边要大 )
170                     mi = g[1][j] - dp[j].v;
171                     tmp = j;
172                 }
173             }
174         }
175         if(mi >= 0) break;    //如果不存在这样的边, 直接退出
176         int x = dp[tmp].x, y = dp[tmp].y;
177
178         flag[1][tmp] = flag[tmp][1] = true;    //加上新找的边
179         flag[x][y] = flag[y][x] = false;    //删掉被替换掉的那条边

```

```

177         ans += mi;
178     }
179     printf("Total miles driven: %d\n", ans);
180
181     return 0;
182 }
183

```

最优比率生成树

```

1  #include<map>
2  #include<cmath>
3  #include<ctime>
4  #include<queue>
5  #include<cstdio>
6  #include<vector>
7  #include<bitset>
8  #include<cstring>
9  #include<iostream>
10 #include<algorithm>
11 #define ll long long
12 #define mod 1000000009
13 #define inf 1000000000
14 #define eps 1e-8
15 using namespace std;
16 int n,cnt;
17 int x[1005],y[1005],z[1005],last[1005];
18 double d[1005],mp[1005][1005],ans;
19 bool vis[1005];
20 void prim(){
21     for(int i=1;i<=n;i++){
22         d[i]=inf;vis[i]=0;
23     }
24     d[1]=0;
25     for(int i=1;i<=n;i++){
26         int now=0;d[now]=inf;
27         for(int j=1;j<=n;j++)if(d[j]<d[now]&&!vis[j])now=j;
28         ans+=d[now];vis[now]=1;
29         for(int j=1;j<=n;j++)
30             if(mp[now][j]<d[j]&&!vis[j])
31                 d[j]=mp[now][j];
32     }
33 }
34 double sqr(double x){
35     return x*x;
36 }
37 double dis(int a,int b){
38     return sqrt(sqr(x[a]-x[b])+sqr(y[a]-y[b]));
39 }

```

```

40 void cal(double mid){
41     ans=0;
42     for(int i=1;i<=n;i++)
43         for(int j=i+1;j<=n;j++)
44             mp[i][j]=mp[j][i]=abs(z[i]-z[j])-mid*dis(i,j);
45     prim();
46 }
47 int main(){
48     while(scanf("%d",&n)){
49         if(n==0)break;
50         for(int i=1;i<=n;i++)
51             scanf("%d%d%d",&x[i],&y[i],&z[i]);
52         double l=0,r=1000;
53         for(int i=1;i<=30;i++)
54             {
55                 double mid=(l+r)/2;
56                 cal(mid);
57                 if(ans<0)r=mid;
58                 else l=mid;
59             }
60         printf("%.3f\n",l);
61     }
62     return 0;
63 }

```

数学

常用公式

积性函数

$\sigma_k(n) = \sum_{d|n} d^k$ 表示 n 的约数的 k 次幂和

$$\sigma_k(n) = \prod_{i=1}^k \frac{(p_i^{a_i+1})^k - 1}{p_i^k - 1}$$

$$\varphi(n) = \sum_{i=1}^n [(n, i) = 1] = \prod_{i=1}^k (1 - \frac{1}{p_i})$$

$$\varphi(p^k) = (p-1)p^{k-1}$$

$$\sum_{d|n} \varphi(d) = n \rightarrow \varphi(n) = n - \sum_{d|n, d < n}$$

$n \geq 2$ 时 $\varphi(n)$ 为偶数

$$\mu(n) = \begin{cases} 0 & \text{有平方因子} \\ (-1)^t & n = \prod_{i=1}^t p_i \end{cases}$$

$[n=1] = \sum_{d|n} \mu(d)$ 排列组合后二项式定理转换即可证明

$n = \sum_{d|n} \varphi(d)$ 将 $\frac{i}{n} (1 \leq i \leq n)$ 化为最简分数统计个数即可证明

莫比乌斯反演

$$\begin{aligned}F(n) = \sum_{d|n} f(d) &\Rightarrow f(n) = \sum_{d|n} \mu(d) * F\left(\frac{n}{d}\right) \\F(n) = \sum_{n|d} f(d) &\Rightarrow f(n) = \sum_{n|d} \mu\left(\frac{n}{d}\right) * F(d) \\f(n) = \sum_{d|n} \phi(d) &\Rightarrow \phi(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) = \sum_{d|n} \mu(d) \frac{n}{d}\end{aligned}$$

常用等式

不知道有什么用

$$\begin{aligned}\sum_{d|N} \phi(d) &= N \\ \sum_{i \leq N} i * [(i, N) = 1] &= \frac{N * \phi(N)}{2} \\ \sum_{d|N} \frac{\mu(d)}{d} &= \frac{\phi(N)}{N}\end{aligned}$$

常用代换

$$\sum_{d|N} \mu(d) = [N = 1]$$

考虑每个数的贡献

$$\sum_{i \leq N} \lfloor \frac{N}{i} \rfloor = \sum_{i \leq N} d(i)$$

SG 函数

```
1 #define MAX 150 //最大的步数
2
3 int step[MAX], sg[10500], steps; //使用前应将sg初始化为-1
4 //step:所有可能的步数,要求从小到大排序
5 //steps:step的大小
6 //sg:存储sg的值
7
8
9 int getsg(int m)
10 {
11     int hashes[MAX] = {0};
12     int i;
13     for (i = 0; i < steps; i++)
14     {
15         if (m - step[i] < 0) {
16             break;
17         }
18         if (sg[m - step[i]] == -1) {
19             sg[m - step[i]] = getsg(m - step[i]);
20         }
21         hashes[sg[m - step[i]]] = 1;
22     }
23     for (i = 0;; i++) {
24         if (hashes[i] == 0) {
25             return i;
26         }
27     }
```

```

27     }
28 }
29
30 /*
31 Array(存储可以走的步数, Array[0]表示可以有多少种走法)
32 Array[] 需要从小到大排序
33 1.可选步数为1-m的连续整数, 直接取模即可, SG(x)=x%(m+1);
34 2.可选步数为任意步, SG(x) = x;
35 3.可选步数为一系列不连续的数, 用GetSG(计算)
36 */
37 //获取sg表
38 int SG[MAX], hashes[MAX];
39
40 void init(int Array[], int n)
41 {
42     int i, j;
43     memset(SG, 0, sizeof(SG));
44     for (i = 0; i <= n; i++)
45     {
46         memset(hashes, 0, sizeof(hashes));
47         for (j = 1; j <= Array[0]; j++)
48         {
49             if (i < Array[j]) {
50                 break;
51             }
52             hashes[SG[i - Array[j]]] = 1;
53         }
54         for (j = 0; j <= n; j++)
55         {
56             if (hashes[j] == 0)
57             {
58                 SG[i] = j;
59                 break;
60             }
61         }
62     }
63 }

```

矩阵乘法快速幂

```

1  /*
2  MAIN为矩阵大小
3  MOD为模数
4  调用pamt(a,k)返回a^k
5  */
6  struct mat{
7      int n, m;
8      int c[MAIN][MAIN];
9  };

```



```

10 mat cheng(const mat &a, const mat &b){
11     mat w;
12     SET(w.c,0);
13     w.n = a.n, w.m = b.m;
14     Rep(i,a.n)Rep(j,b.m)Rep(k,a.m){
15         w.c[i][j] += (ll)a.c[i][k] * b.c[k][j] % MOD;
16         if(w.c[i][j]>MOD)w.c[i][j]-=MOD;
17     }
18     return w;
19 }
20 mat pmat(mat a, ll k){
21     mat i;
22     i.n = i.m = MATN;
23     SET(i.c,0);
24     Rep(i,MATN)
25         i.c[i][i] = 1;
26     while(k){
27         if(k&1)
28             i=cheng(i,a);
29         a=cheng(a,a);
30         k>>=1;
31     }
32     return i;
33 }

```

线性基

```

1  /*
2     求一条从1到n的路径,使得路径上的边的异或和最大。
3  */
4  #include <cstdio>
5  #include <algorithm>
6  using namespace std;
7  #define N 50001
8  #define M 100001
9  struct E
10 {
11     int u, v, next;
12     long long w;
13     E(int _u = 0, int _v = 0, int _next = 0, long long _w = 0){u = _u, v =
        _v, next = _next, w = _w;}
14 }G[M<1];
15 int cnt, point[N], n, m;
16 char c;
17 template<class T>
18 inline void read(T &x)
19 {
20     T opt(1);

```

```

21     for (c = getchar(); c > '9' || c < '0'; c = getchar()) if (c == '-') opt =
22         -1;
23     for (x = 0; c >= '0' && c <= '9'; c = getchar()) x = (x << 3) + (x << 1) +
24         c - '0';
25     x *= opt;
26 }
27 bool vis[N];
28 long long dis[N];
29 long long a[M<1];
30 int Gauss()
31 {
32     int i, j(0), k;
33     for (i = 63; i >= 0; --i)
34     {
35         for (k = j+1; k <= n; ++k)
36             if ((a[k] >> i) & 1) break;
37         if (k > n) continue;
38         swap(a[k], a[j+1]);
39         for (k = 1; k <= n; ++k)
40             if (j+1 != k && ((a[k] >> i) & 1))
41                 a[k] ^= a[j+1];
42         j++;
43     }
44     return j;
45 } inline void dfs(int u)
46 {
47     vis[u] = 1;
48     int i, v;
49     for (i = point[u]; i; i = G[i].next)
50     {
51         v = G[i].v;
52         if (vis[v])
53             a[++m] = dis[u] ^ dis[v] ^ G[i].w;
54         else
55         {
56             dis[v] = dis[u] ^ G[i].w;
57             dfs(v);
58         }
59     }
60 }
61 int main()
62 {
63     read(n), read(m);
64     int i, j, u, v, k;
65     long long w, ans;
66     for (i = 1; i <= m; ++i)
67     {
68         read(u), read(v), read(w);
69         G[++cnt] = E(u, v, point[u], w), point[u] = cnt;
70         G[++cnt] = E(v, u, point[v], w), point[v] = cnt;
71     }

```

```

70     m = 0;
71     dfs(1);
72     ans = dis[n];
73     n = m;
74     k = Gauss();
75     for (i = k; i; —i)
76         ans = max(ans, ans ^ a[i]);
77     printf("%lld\n", ans);
78     return 0;
79 }

```

线性筛

```

1  /*
2     is 是不是质数
3     phi 欧拉函数
4     mu 莫比乌斯函数
5     minp 最小质因子
6     mina 最小质因子次数
7     d 约数个数
8  */
9  int prime[N];
10 int size;
11 int is[N];
12 int phi[N]; // 欧拉函数
13 int mu[N]; // 莫比乌斯函数
14 int minp[N]; // 最小质因子
15 int mina[N]; // 最小质因子次数
16 int d[N]; // 约数个数
17 void getprime(int list){
18     SET(is, 1);
19     mu[1] = 1;
20     phi[1] = 1;
21     is[1] = 0;
22     repab(i, 2, list){
23         if(is[i]){
24             prime[++size] = i;
25             phi[i] = i - 1;
26             mu[i] = -1;
27             minp[i] = i;
28             mina[i] = 1;
29             d[i] = 2;
30         }
31         rep(j, size){
32             if(i * prime[j] > list)
33                 break;
34             is[i * prime[j]] = 0;
35             minp[i * prime[j]] = prime[j];
36             if(i % prime[j] == 0){

```

```

37         mu[i*prime[j]] = 0;
38         phi[i*prime[j]] = phi[i] * prime[j];
39         mina[i*prime[j]] = mina[i]+1;
40         d[i*prime[j]] = d[i]/(mina[i]+1)*(mina[i]+2);
41         break;
42     }else{
43         phi[i*prime[j]] = phi[i] * (prime[j] - 1);
44         mu[i*prime[j]] = -mu[i];
45         mina[i*prime[j]] = 1;
46         d[i*prime[j]] = d[i]*d[prime[j]];
47     }
48 }
49 }
50 }

```

整数卷积 NTT

```

1  /*
2   计算形式为 $a[n] = \sum b[n-i] * c[i]$ 的卷积,结果存在c中
3   下标从0开始
4   调用juanjie(n,b,c)
5   P为模数
6   G是P的原根
7  */
8  const ll P=998244353;
9  const ll G=3;
10 void change(ll y[],int n){
11     int b=n>>1,s=n-1;
12     for(int i=1,j=n>>1;i<s;i++){
13         if(i<j)swap(y[i],y[j]);
14         int k=b;
15         while(j>=k){
16             j-=k;
17             k>>=1;
18         }
19         j+=k;
20     }
21 }
22 void NTT_(ll y[],int len,int on){
23     change(y,len);
24     for(int h=2;h<=len;h<<=1){
25         ll wh=powm(G,(P-1)/h,P);
26         if(on<0)wh=powm(wh,P-2,P);
27         for(int i=0;i<len;i+=h){
28             ll w=1;
29             int r=h>>1;
30             for(int k=i,s=r+i;k<s;k++){
31                 ll u=y[k];
32                 ll t=w*y[k+r]%P;

```

```

33         y[k]=u+t;
34         if(y[k]>=P)y[k]-=P;
35         y[k+r]=u-t;
36         if(y[k+r]<0)y[k+r]+=P;
37         w=w*wt%P;
38     }
39 }
40 }
41 if(on<0){
42     ll I=powm((ll)len,P-2,P);
43     Rep(i,len)y[i]=y[i]*I%P;
44 }
45 }
46 void juanji(int n, ll *b, ll *c){
47     int len=1;
48     while(len<(n<<1))len<<=1;
49     Repab(i,n,len)c[i]= b[i] = 0;
50     NTT_(b,len,1);
51     NTT_(c,len,1);
52     Rep(i,len)
53         c[i]= c[i]*b[i]%P;
54     NTT_(c,len,-1);
55 }

```

中国剩余定理

```

1  /*
2   合并ai在模mi下的结果为模m_0*m_1*...*m_n-1
3  */
4  inline int exgcd(int a, int b, int &x, int &y){
5      if(!b){
6          x = 1, y = 0;
7          return a;
8      }
9      else{
10         int d = exgcd(b, a % b, x, y), t = x;
11         x = y, y = t - a / b * y;
12         return d;
13     }
14 }
15 inline int inv(int a, int p){
16     int d, x, y;
17     d = exgcd(a, p, x, y);
18     return d == 1 ? (x + p) % p : -1;
19 }
20 int china(int n,int *a,int *m){
21     int __M=MOD-1, d, x = 0, y;
22     for(int i = 0;i < n; ++i){
23         int w = __M / m[i];

```

```

24         d = exgcd(m[i], w, d, y);
25         x = (x + ((long long)y*w%__M)*(long long)a[i]%__M)%__M;
26     }
27     while(x <= 0)
28         x += __M;
29     return x;
30 }

```

字符串

AC 自动机

```

1  /// AC自动机.
2
3  /// mxn: 自动机的节点池子大小.
4  const int mxn = 105000;
5
6  /// ct: 字符集大小.
7  const int cst = 26;
8
9  /// 重新初始化:
10 node*pt = pool;
11
12 //////////////////////////////////////
13
14 struct node
15 {
16     node*s[cst];    // Trie 转移边.
17     node*trans[cst]; // 自动机转移边.
18     node*f;         // Fail 指针.
19     char v;          // 当前节点代表字符(父节点指向自己的边代表的字符).
20     bool leaf;       // 是否是某个字符串的终点. 注意该值为true不一定是叶子.
21     node() { } // 保留初始化.
22 }
23 pool[mxn]; node*pt=pool;
24 node* newnode() { memset(pt, 0, sizeof(node)); return pt++; }
25
26 /// 递推队列.
27 node*qc[mxn];
28 node*qf[mxn];
29 int qh,qt;
30
31 struct Trie
32 {
33     node*root;
34     Trie(){ root = newnode(); root->v = '*' - 'a'; }
35
36     /// g: 需要插入的字符串; len:长度.

```

```

37 void Insert(char* g, int len)
38 {
39     node*x=root;
40     for(int i=0;i<len;i++)
41     {
42         int v = g[i]- 'a';
43         if(x->s[v] == NULL)
44         {
45             x->s[v] = newnode();
46             x->s[v]->v = v;
47         }
48         x = x->s[v];
49     }
50     x->leaf = true;
51 }
52
53 // 在所有字符串插入之后执行.
54 // BFS递推, qc[i]表示队中节点指针, qf表示队中对应节点的fail指针.
55 void Construct()
56 {
57     node*x = root;
58     qh = qt = 0;
59     for(int i=0; i<cst; i++) if(x->s[i])
60     {
61         x->s[i]->f = root;
62         for(int j=0; j<cst; j++) if(x->s[i]->s[j])
63         { qc[qt] = x->s[i]->s[j]; qf[qt]=root; qt++; }
64     }
65
66     while(qh != qt)
67     {
68         node*cur = qc[qh];
69         node*fp = qf[qh];
70         qh++;
71
72         while(fp != root && fp->s[cur->v] == NULL) fp = fp->f;
73         if(fp->s[cur->v]) fp = fp->s[cur->v];
74         cur->f = fp;
75
76         for(int i=0; i<cst; i++)
77             if(cur->s[i]) { qc[qt] = cur->s[i]; qf[qt] = fp; qt++; }
78     }
79 }
80
81 // 拿到转移点.
82 // 暴力判定.
83 node* GetTrans(node*x, int v)
84 {
85     while(x != root && x->s[v] == NULL) x = x->f;
86     if(x->s[v]) x = x->s[v];
87     return x;

```

```

88     }
89
90     // 拿到转移点.
91     // 记忆化搜索.
92     node* GetTrans(node*x, int v)
93     {
94         if(x->s[v]) return x->trans[v] = x->s[v];
95
96         if(x->trans[v] == NULL)
97         {
98             if(x == root) return root;
99             return x->trans[v] = GetTrans(x->f, v);
100         }
101
102         return x->trans[v];
103     }
104 };

```

KMP

```

1  //KMP算法
2  //查找成功则返回所在位置(int),否则返回-1.
3
4  #define MAXM 100000000 //字符串最大长度
5
6  void getNext(char *p, char *next)
7  {
8      int j = 0;
9      int k = -1;
10     next[0] = -1;
11     while (j < n)
12     {
13         if (k == -1 || p[j] == p[k])
14         {
15             j++;
16             k++;
17             next[j] = k;
18         }
19         else
20             k = next[k];
21     }
22 }
23
24 int KMP(char *s, char *p, int m, int n) //查找成功则返回所在位置(int),否则返回-1.
25 { //s为文本串,p为模式串;m为文本串长度,n为模式串长度.
26     char next[MAXM];
27     int i = 0;

```



```

28     int j = 0;
29     getNext(p, next);
30     while (i < m)
31     {
32         if (j == -1 || s[i] == p[j])
33         {
34             i++;
35             j++;
36         }
37         else
38             j = next[j];
39         if (j == n)
40             return i - n + 1;
41     }
42     return -1;
43 }

```

Manacher

```

1  #define MAXM 20001
2  //返回回文串的最大值
3  //MAXM至少应为输入字符串长度的两倍+1
4
5  int p[MAXM];
6  char s[MAXM];
7
8  int manacher(string str) {
9      memset(p, 0, sizeof(p));
10     int len = str.size();
11     int k;
12     for (k = 0; k < len; k++) {
13         s[2 * k] = '#';
14         s[2 * k + 1] = str[k];
15     }
16     s[2 * k] = '#';
17     s[2 * k + 1] = '\0';
18     len = strlen(s);
19     int mx = 0;
20     int id = 0;
21     for (int i = 0; i < len; ++i) {
22         if (i < mx) {
23             p[i] = min(p[2 * id - i], mx - i);
24         }
25         else {
26             p[i] = 1;
27         }
28         for (; s[i - p[i]] == s[i + p[i]] && s[i - p[i]] != '\0' && s[i + p[
29             i]] != '\0'; ) {

```

```

30     }
31     if ( p[i] + i > mx ) {
32         mx = p[i] + i;
33         id = i;
34     }
35 }
36 int res = 0;
37 for (int i = 0; i < len; ++i) {
38     res = max(res, p[i]);
39 }
40 return res - 1;
41 }

```

Trie 树

```

1  #define CHAR_SIZE 26          //字符种类数
2  #define MAX_NODE_SIZE 10000  //最大节点数
3
4  inline int getCharID(char a) { //返回a在子数组中的编号
5      return a - 'a';
6  }
7
8  struct Trie
9  {
10     int num; //记录多少单词途径该节点, 即多少单词拥有以该节点为末尾的前缀
11     bool terminal; //若terminal==true, 该节点没有后续节点
12     int count; //记录单词的出现次数, 此节点即一个完整单词的末尾字母
13     struct Trie *son[CHAR_SIZE]; //后续节点
14 };
15
16 struct Trie trie_arr[MAX_NODE_SIZE];
17 int trie_arr_point=0;
18
19 Trie *NewTrie()
20 {
21     Trie *temp=&trie_arr[trie_arr_point++];
22     temp->num=1;
23     temp->terminal=false;
24     temp->count=0;
25     for (int i=0; i<sonnum; ++i) temp->son[i]=NULL;
26     return temp;
27 }
28
29 //插入新词, root: 树根, s: 新词, len: 新词长度
30 void Insert(Trie *root, char *s, int len)
31 {
32     Trie *temp=root;
33     for (int i=0; i<len; ++i)
34     {

```

```

35         if(temp->son[getCharID(s[i])]==NULL)temp->son[getCharID(s[i])]=
            NewTrie();
36         else {temp->son[getCharID(s[i])]->num++;temp->terminal=false;}
37         temp=temp->son[getCharID(s[i])];
38     }
39     temp->terminal=true;
40     temp->count++;
41 }
42 //删除整棵树
43 void Delete()
44 {
45     memset(trie_arr,0,trie_arr_point*sizeof(Trie));
46     trie_arr_point=0;
47 }
48 //查找单词在字典树中的末尾节点.root:树根,s:单词,len:单词长度
49 Trie* Find(Trie *root,char *s,int len)
50 {
51     Trie *temp=root;
52     for(int i=0;i<len;++i)
53         if(temp->son[getCharID(s[i])]!=NULL)temp=temp->son[getCharID(s[i])];
54     else return NULL;
55     return temp;
56 }

```

后缀数组-DC3

```

1 //dc3函数:s为输入的字符串,sa为结果数组,slen为s长度,m为字符串中字符的最大值+1
2 //s及sa数组的大小应为字符串大小的3倍.
3
4 #define MAXN 100000 //字符串长度
5
6 #define F(x) ((x)/3+((x)%3==1?0:tb))
7 #define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
8
9 int wa[MAXN], wb[MAXN], wv[MAXN], ws[MAXN];
10
11 int c0(int *s, int a, int b)
12 {
13     return s[a] == s[b] && s[a+1] == s[b+1] && s[a+2] == s[b+2];
14 }
15
16 int c12(int k, int *s, int a, int b)
17 {
18     if (k == 2) return s[a] < s[b] || s[a] == s[b] && c12(1, s, a+1, b+
19         1);
20     else return s[a] < s[b] || s[a] == s[b] && wv[a+1] < wv[b+1];
21 }
22 void sort(int *s, int *a, int *b, int slen, int m)

```

```

23 {
24     int i;
25     for (i = 0; i < slen; i++) wv[i] = s[a[i]];
26     for (i = 0; i < m; i++) ws[i] = 0;
27     for (i = 0; i < slen; i++) ws[wv[i]]++;
28     for (i = 1; i < m; i++) ws[i] += ws[i - 1];
29     for (i = slen - 1; i >= 0; i--) b[--ws[wv[i]]] = a[i];
30     return;
31 }
32
33 void dc3(int *s, int *sa, int slen, int m)
34 {
35     int i, j, *rn = s + slen, *san = sa + slen, ta = 0, tb = (slen + 1) / 3,
36         tbc = 0, p;
37     s[slen] = s[slen + 1] = 0;
38     for (i = 0; i < slen; i++) if (i % 3 != 0) wa[tbc++] = i;
39     sort(s + 2, wa, wb, tbc, m);
40     sort(s + 1, wb, wa, tbc, m);
41     sort(s, wa, wb, tbc, m);
42     for (p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++)
43         rn[F(wb[i])] = c0(s, wb[i - 1], wb[i]) ? p - 1 : p++;
44     if (p < tbc) dc3(rn, san, tbc, p);
45     else for (i = 0; i < tbc; i++) san[rn[i]] = i;
46     for (i = 0; i < tbc; i++) if (san[i] < tb) wb[ta++] = san[i] * 3;
47     if (slen % 3 == 1) wb[ta++] = slen - 1;
48     sort(s, wb, wa, ta, m);
49     for (i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;
50     for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++)
51         sa[p] = c12(wb[j] % 3, s, wa[i], wb[j]) ? wa[i++] : wb[j++];
52     for (; i < ta; p++) sa[p] = wa[i++];
53     for (; j < tbc; p++) sa[p] = wb[j++];
54     return;
55 }

```

后缀数组-倍增法

```

1 #define MAXN 100000 //字符串长度
2
3 //da函数:s为输入的字符串,sa为结果数组,slen为s长度,m为字符串中字符的最大值+1
4 //调用前应将s[slen]设为0,因此调用时slen为s长度+1
5
6 //calHeight函数:返回sa中排名相邻的两个后缀的最长公共前缀
7
8 int cmp(int *s, int a, int b, int l) {
9     return (s[a] == s[b]) && (s[a + l] == s[b + l]);
10 }
11
12 int wa[MAXN], wb[MAXN], ws[MAXN], wv[MAXN];
13 void da(int *s, int *sa, int slen, int m) {

```

```

14     int i, j, p, *x = wa, *y = wb, *t;
15     for (i = 0; i < m; i++) ws[i] = 0;
16     for (i = 0; i < slen; i++) ws[x[i]] = s[i]++;
17     for (i = 1; i < m; i++) ws[i] += ws[i - 1];
18     for (i = slen - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
19     for (j = 1, p = 1; p < slen; j *= 2, m = p)
20     {
21         for (p = 0, i = slen - j; i < slen; i++) y[p++] = i;
22         for (i = 0; i < slen; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
23         for (i = 0; i < slen; i++) wv[i] = x[y[i]];
24         for (i = 0; i < m; i++) ws[i] = 0;
25         for (i = 0; i < slen; i++) ws[wv[i]]++;
26         for (i = 1; i < m; i++) ws[i] += ws[i - 1];
27         for (i = slen - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
28         for (t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < slen; i++)
29             x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
30     }
31 }
32
33 //-----
34 int rank[MAXN], height[MAXN];
35 void calHeight(int *s, int *sa, int slen) {
36     int i, j, k = 0;
37     for (i = 1; i <= slen; i++) rank[sa[i]] = i;
38     for (i = 0; i < slen; height[rank[i++]] = k )
39         for (k ? k-- : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; k++);
40 }
41 //-----

```

后缀自动机

```

1  /*
2     求多个串的LCS
3  */
4  #include <cstdio>
5  #include <cstring>
6  #include <algorithm>
7  using namespace std;
8  #define N 100001
9  struct node
10 {
11     node *suf, *s[26], *next;
12     int val, w[11];
13 }*r, *l, T[N<<1+1];
14 node *point[N];
15 char str[N];
16 int n, len, k, tot;
17 inline void add(int w)
18 {

```

```

19     node *p = l, *np = &T[tot++];
20     np->val = p->val+1;
21     np->next = point[np->val], point[np->val] = np;
22     while (p && !p->s[w])
23         p->s[w] = np, p = p->suf;
24     if (!p)
25         np->suf = r;
26     else
27     {
28         node *q = p->s[w];
29         if (p->val+1 == q->val)
30             np->suf = q;
31         else
32         {
33             node *nq = &T[tot++];
34             memcpy(nq->s, q->s, sizeof q->s);
35             nq->val = p->val+1;
36             nq->next = point[p->val+1], point[p->val+1] = nq;
37             nq->suf = q->suf;
38             q->suf = nq;
39             np->suf = nq;
40             while (p && p->s[w] == q)
41                 p->s[w] = nq, p = p->suf;
42         }
43     }
44     l = np;
45 }
46 int main()
47 {
48     freopen("a.in", "r", stdin);
49     int i, j, now, L, res, ans(0), w;
50     node *p;
51     r = l = &T[tot++];
52     r->next = point[0], point[0] = r;
53     scanf("%s", str);
54     L = strlen(str);
55     for (i = 0; i < L; ++i)
56         add(str[i] - 'a');
57     for (tot = 1; scanf("%s", str) != EOF; ++tot)
58     {
59         len = strlen(str);
60         p = r, now = 0;
61         for (j = 0; j < len; ++j)
62         {
63             w = str[j] - 'a';
64             if (p->s[w])
65                 p = p->s[w], p->w[tot] = max(p->w[tot], ++now);
66             else
67             {
68                 while (p && !p->s[w])
69                     p = p->suf;

```

```

70         if (!p)
71             p = r, now = 0;
72         else
73             now = p->val+1, p = p->s[w], p->w[tot] = max(p->w[tot],
74                 now);
75     }
76 }
77 for (i = L; i >= 0; --i)
78     for (node *p = point[i]; p; p = p->next)
79     {
80         res = p->val;
81         for (j = 1; j < tot; ++j)
82         {
83             res = min(p->w[j], res);
84             if (p->suf)
85                 p->suf->w[j] = max(p->suf->w[j], p->w[j]);
86         }
87         ans = max(ans, res);
88     }
89 printf("%d\n", ans);
90 return 0;
91 }

```

扩展 KMP

```

1  //使用getExtend获取extend数组(s[i]...s[n-1]与t的最长公共前缀的长度)
2  //s,t,slen,tlen,分别为对应字符串及其长度.
3  //next数组返回t[i]...t[m-1]与t的最长公共前缀长度,调用时需要提前开辟空间
4  void getNext(char* t, int tlen, int* next)
5  {
6      next[0] = tlen;
7      int a;
8      int p;
9      for (int i = 1, j = -1; i < tlen; i++, j--)
10     {
11         if (j < 0 || i + next[i - a] >= p)
12         {
13             if (j < 0) {
14                 p = i;
15                 j = 0;
16             }
17             while (p < tlen && t[p] == t[j]) {
18                 p++;
19                 j++;
20             }
21             next[i] = j;
22             a = i;
23         }
24     }
25 }

```

```

24         else {
25             next[i] = next[i - a];
26         }
27     }
28 }
29
30 void getNext(char* s, int slen, char* t, int tlen, int* extend, int* next)
31 {
32     getNext(t, next);
33     int a;
34     int p;
35
36     for (int i = 0, j = -1; i < slen; i++, j--)
37     {
38         if (j < 0 || i + next[i - a] >= p)
39         {
40             if (j < 0) {
41                 p = i, j = 0;
42             }
43             while (p < slen && j < tlen && s[p] == t[j]) {
44                 p++;
45                 j++;
46             }
47             extend[i] = j;
48             a = i;
49         }
50         else {
51             extend[i] = next[i - a];
52         }
53     }
54 }

```

杂项

测速

```

1  /*
2   * require c++11 support
3   */
4  #include <chrono>
5  using namespace chrono;
6  int main(){
7      auto start = system_clock::now();
8      //do something
9      auto end = system_clock::now();
10     auto duration = duration_cast<microseconds>(end - start);
11     cout << double(duration.count()) * microseconds::period::num /
        microseconds::period::den << endl;

```



```
12 }
```

日期公式

```
1  /*
2      zeller 返回星期几%7
3  */
4  int zeller(int y,int m,int d) {
5      if (m<=2) y--m+=12; int c=y/100; y%=100;
6      int w=((c>>2)-(c<<1)+y+(y>>2)+(13*(m+1)/5)+d-1)%7;
7      if (w<0) w+=7; return(w);
8  }
9  /*
10     用于计算天数
11  */
12 int getId(int y, int m, int d) {
13     if (m < 3) {y --; m += 12;}
14     return 365 * y + y / 4 - y / 100 + y / 400 + (153 * m + 2) / 5 + d;
15 }
```

读入挂

```
1  // BUF_SIZE对应文件大小
2  // 调用read(x)或者x=getint()
3  #define BUF_SIZE 100000
4  bool IOError = 0;
5  inline char nc() { // next char
6      static char buf[BUF_SIZE], *p1 = buf + BUF_SIZE, *pend = buf + BUF_SIZE;
7      if(p1 == pend){
8          p1 = buf;
9          pend = buf + fread(buf, 1, BUF_SIZE, stdin);
10         if(pend == p1){
11             IOError = 1;
12             return -1;
13         }
14     }
15     return *p1++;
16 }
17 inline bool blank(char ch){
18     return ch == ' ' || ch == '\n' || ch == '\r' || ch == '\t';
19 }
20 inline void read(int &x){
21     char ch;
22     int sgn = 1;
23     while(blank(ch = nc()));
24     if(IOError)
25         return;
```

```

26     if(ch=='-')sgn=-1,ch=nc();
27     for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0'
28     );
29     x*=sgn;
30 }
31 inline int getint(){
32     int x=0;
33     char ch;
34     int sgn = 1;
35     while(blank(ch = nc()));
36     if(IOerror)
37         return;
38     if(ch=='-')sgn=-1,ch=nc();
39     for(x = ch - '0'; (ch = nc()) >= '0' && ch <= '9'; x = x * 10 + ch - '0'
40     );
41     x*=sgn;
42     return x;
43 }
44 inline void print(int x){
45     if (x == 0){
46         puts("0");
47         return;
48     }
49     short i, d[101];
50     for (i = 0;x; ++i)
51         d[i] = x % 10, x /= 10;
52     while (i--)
53         putchar(d[i] + '0');
54     puts("");
55 }
56 #undef BUF_SIZE

```

高精度

```

1  #include <cstdio>
2  #include <cstdlib>
3  #include <cstring>
4  #include <cmath>
5
6  #include <iostream>
7  #include <algorithm>
8
9  #include <map>
10 #include <stack>
11
12 typedef long long ll;
13 typedef unsigned int uint;
14 typedef unsigned long long ull;
15 typedef double db;

```

```

16 typedef unsigned char uchar;
17
18 using namespace std;
19 inline bool isnum(char c) { return '0' <= c && c <= '9'; }
20 inline int getint(int x=0) { scanf("%d", &x); return x; }
21 inline ll getll(ll x=0) { scanf("%lld", &x); return x; }
22 double getdb(double x=0) { scanf("%lf",&x); return x; }
23
24 //=====
25
26 /// 大整数模板.
27 /// 这个模板保证把一个数字存成  $v[0]*SYS^0 + v[1]*SYS^1 + \dots$  的形式.
28 /// 支持负数运算.
29 struct bign
30 {
31     static const int SYS = 10; // 多少进制数.
32     static const int SIZE = 200; // 数位数.
33     int v[SIZE]; // 数位, 从0到N从低到高. 注意可能会爆栈, 可以把它换成指针.
34     int len;
35
36     //=====
37     //                                工具函数
38     //=====
39
40     /// 进位和退位整理.
41     void Advance(int const& i)
42     {
43         int k = v[i] / SYS;
44         v[i] %= SYS;
45         if(v[i] < 0) { k--; v[i] += SYS; }
46         v[i+1] += k;
47     }
48
49     /// 进位和退位处理. 注意不会减少len.
50     void Advance()
51     { for(int i=0; i<len; i++) Advance(i); if(v[len] != 0) len++; }
52
53     /// 去除前导0和前导-1.
54     void Strip()
55     {
56         while(len > 0 && v[len-1] == 0) len--;
57         while(len > 0 && v[len-1] == -1 && v[len-1] != 0) { len--; v[len] =
58             0; v[len-1] += 10; }
59
60     bool isNegative() const { return len != 0 && v[len-1] < 0; }
61
62     int& operator [] (int const& k) { return v[k]; }
63
64     //=====
65     //                                构造函数

```

```

66 //=====
67
68 // 初始化为0.
69 bign() { memset(this, 0, sizeof(bign)); }
70
71 // 从整数初始化.
72 bign(ll k)
73 {
74     memset(this, 0, sizeof(bign));
75     while(k != 0) { v[len++] = k % SYS; k /= SYS; }
76     Advance();
77 }
78
79 // 从字符串初始化. 仅十进制. 支持 -0, 0, 正数, 负数. 不支持前导0, 如
    00012, -000, -0101.
80 bign(const char* f)
81 {
82     memset(this, 0, sizeof(bign));
83     if(f[0] == '-')
84     {
85         f++;
86         int l = strlen(f);
87         for(int i=l-1; i>=0; i--) v[len++] = -(f[i] - '0');
88         Advance();
89         if(len == 1 && v[len-1] == 0) len = 0;
90     }
91     else
92     {
93         int l = strlen(f);
94         for(int i=l-1; i>=0; i--) v[len++] = f[i] - '0';
95         if(len == 1 && v[0] == 0) len--;
96     }
97 }
98
99 // 拷贝构造函数.
100 bign(bign const& f) { memcpy(this, &f, sizeof(bign)); }
101
102 // 拷贝函数.
103 bign operator=(bign const& f)
104 {
105     memcpy(this, &f, sizeof(bign));
106     return *this;
107 }
108
109 //=====
110 //                                比较大小
111 //=====
112
113 bool operator==(bign const& f) const
114 {
115     if(len != f.len) return false;

```

```

116         for(int i=0; i<len; i++) if(v[i] != f.v[i]) return false;
117         return true;
118     }
119
120     bool operator<(bign const& f) const
121     {
122         if(isNegative() && !f.isNegative()) return true;
123         if(!isNegative() && f.isNegative()) return false;
124         if(isNegative() && f.isNegative())
125         {
126             if(len != f.len) return len > f.len;
127             for(int i=len-1; i>=0; i--) if(v[i] != f.v[i]) return v[i] > f.v
128                 [i];
129             return false;
130         }
131         if(len != f.len) return len < f.len;
132         for(int i=len-1; i>=0; i--) if(v[i] != f.v[i]) return v[i] < f.v[i];
133         return false;
134     }
135
136     bool operator>(bign const& f) const { return f < *this; }
137     bool operator<=(bign const& f) const { return !(*this > f); }
138     bool operator>=(bign const& f) const { return !(*this < f); }
139
140     //=====
141     //                      运算
142     //=====
143
144     bign operator-() const
145     {
146         bign c = *this;
147         for(int i=c.len-1; i>=0; i--) { c[i] = -c[i]; }
148         c.Advance();
149         c.Strip();
150         return c;
151     }
152
153     bign operator+(bign const& f) const
154     {
155         bign c;
156         c.len = max(len, f.len);
157         for(int i=0; i<c.len; i++) c[i] = v[i] + f.v[i];
158         c.Advance();
159         c.Strip();
160         return c;
161     }
162
163     bign operator-(bign const& f) const { return *this + (-f); }
164
165     bign operator*(int const& k) const

```

```

166     {
167         bign c;
168         c.len = len;
169         for(int i=0; i<len; i++) c.v[i] = v[i] * k;
170         c.len += 10; // 这个乘数需要设置成比 log(SYS, max(k)) 大.
171         c.Advance();
172         c.Strip();
173         return c;
174     }
175
176     bign operator*(bign const& f) const
177     {
178         if(isNegative() && f.isNegative()) return ((-*this) * (-f));
179         if(isNegative()) return - ((*this) * f);
180         if(f.isNegative()) return - (*this * (-f));
181         bign c;
182         c.len = len + f.len;
183         for(int i=0; i<len; i++)
184         {
185             for(int j=0; j<f.len; j++) c[i+j] += v[i] * f.v[j];
186             c.Advance();
187         }
188         c.Strip();
189         return c;
190     }
191
192     int operator%(int const& k) const
193     {
194         int res = 0;
195         for(int i=len-1; i>=0; i--) (res = res * SYS + v[i]) %= k;
196         return res;
197     }
198
199     //=====
200     //                               输入输出
201     //=====
202
203     bign Out(const char* c = "\n") const
204     {
205         if(len == 0 || (len == 1 && v[0] == 0)) { printf("0%s", c); return *
                this; }
206         if(v[len-1] >= 0)
207         {
208             for(int i=len-1; i>=0; i--) printf("%d", v[i]);
209             printf("%s", c);
210         }
211         else
212         {
213             printf("-");
214             (-*this).Out(c);
215         }

```

```

216         return *this;
217     }
218
219     bign TestOut(const char* c = "\n", int const& sz = 0) const
220     {
221         printf("[(%d)␣", len);
222         if(sz == 0) for(int i=0; i<len; i++) printf("%d␣", v[i]);
223         else for(int i=0; i<sz; i++) printf("%d␣", v[i]);
224         printf("]\n");
225         Out("");
226         printf("%s", c);
227         return *this;
228     }
229
230 };

```

康托展开与逆展开

```

1  /// 康托展开.
2
3  /// 从一个排列映射到排列的rank.
4  /// power : 阶乘数组.
5
6  //////////////////////////////////////
7
8  int power[21];
9
10 /// 康托展开, 排名从0开始.
11 /// 输入为字符串, 其中的字符根据ascii码比较大小.
12 /// 可以将该字符串替换成其它线性集合中的元素的排列.
13 int Cantor(const char* c, int len)
14 {
15     int res = 0;
16     for(int i=0; i<len; i++)
17     {
18         int rank = 0;
19         for(int j=i; j<len; j++) if(c[j] < c[i]) rank++;
20         res += rank * power[len - i - 1];
21     }
22     return res;
23 }
24
25 bool cused[21]; // 该数组大小应为字符集的大小.
26 /// 逆康托展开, 排名从0开始.
27 /// 输出排名为rank的, 长度为len的排列.
28 void RevCantor(int rank, char* c, int len)
29 {
30     for(int i=0; i<len; i++) cused[i] = false;
31     for(int i=0; i<len; i++)

```

```

32     {
33         int cnt = rank / power[len - i - 1];
34         rank %= power[len - i - 1];
35         cnt++;
36         int num = 0;
37         while(true)
38         {
39             if(!cused[num]) cnt--;
40             if(cnt == 0) break;
41             num++;
42         }
43         cused[num] = true;
44         c[i] = num + 'a'; // 输出字符串，从a开始。
45     }
46 }
47
48 /// 阶乘数组初始化。
49 int main()
50 {
51     power[0] = power[1] = 1;
52     for(int i=0; i<20; i++) power[i] = i * power[i-1];
53     ...
54 }

```

快速乘

```

1 inline ll mul(ll a, ll b){
2     ll d=(ll)floor(a*(double)b/M+0.5);
3     ll ret=a*b-d*M;
4     if(ret<0)ret+=M;
5     return ret;
6 }

```

模拟退火

```

1 /// 模拟退火。
2 /// 可能需要魔法调参。慎用！
3
4 /// Tbegin: 退火起始温度。
5 /// Tend: 退火终止温度。
6 /// rate: 退火比率。
7 /// 退火公式: rand_range(0, 1) > exp(dist / T), 其中 dist 为计算出的优化增量
8
9 //////////////////////////////////////
10
11 srand(11212);

```



```

12 db Tbegin = 1e2;
13 db Tend = 1e-6;
14 db T = Tbegin;
15 db rate = 0.99995;
16 int tcnt = 0;
17 point mvbase = point(0.01, 0.01);
18 point curp = p[1];
19 db curmax = GetIntArea(curp);
20 while(T >= Tend)
21 {
22     // 生成一个新的解.
23     point nxtp = curp + point(
24         (randdb() - 0.5) * 2.0 * mvbase.x * T,
25         (randdb() - 0.5) * 2.0 * mvbase.y * T);
26
27     // 计算这个解的价值.
28     db v = GetIntArea(nxtp);
29
30     // 算出距离当前最优解有多远.
31     db dist = v - curmax;
32     if(dist > eps || (dist < -eps && randdb() > exp(dist / T)))
33     {
34         // 更新方案和答案.
35         curmax = v;
36         curp = nxtp;
37         tcnt++;
38     }
39
40     T *= rate;
41 }

```

常用概念

映射

[injective] or [one-to-one] 函数值不重复

[surjective] or [onto] 值域都被取到

[bijective] or [one-to-one correspondence] 一一对应

反演

反演中心 O , 反演半径 r , 点 p 的反演点 p' 满足 $|OP||OP'| = r^2$

不经过反演中心的直线, 反形为经过反演中心的圆

不经过反演中心的圆, 反形为圆, 反演中心为这两个互为反形的圆的位似中心

弦图

设 $next(v)$ 表示 $N(v)$ 中最前的点. 令 w^* 表示所有满足 $A \in B$ 的 w 中最后的一个点, 判断 $v \cup N(v)$ 是否为极大团, 只需判断是否存在一个 $w \in w^*$, 满足 $Next(w) = v$ 且 $|N(v)| + 1 \leq |N(w)|$ 即可.

五边形数

$$\prod_{n=1}^{\infty} (1 - x^n) = \sum_{n=0}^{\infty} (-1)^n (1 - x^{2n+1}) x^{n(3n+1)/2}$$

重心

半径为 r , 圆心角为 θ 的扇形重心与圆心的距离为 $\frac{4r \sin(\theta/2)}{3\theta}$

半径为 r , 圆心角为 θ 的圆弧重心与圆心的距离为 $\frac{4r \sin^3(\theta/2)}{3(\theta - \sin(\theta))}$

第二类 Bernoulli number

$$B_m = 1 - \sum_{k=0}^{m-1} \binom{m}{k} \frac{B_k}{m-k+1}$$

$$S_m(n) = \sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k n^{m+1-k}$$

Catalan 数

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

前 20 项: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190

Stirling 数

第一类: n 个元素的项目分作 k 个环排列的方法数目

$$s(n, k) = (-1)^{n+k} |s(n, k)|$$

$$|s(n, 0)| = 0$$

$$|s(1, 1)| = 1$$

$$|s(n, k)| = |s(n-1, k-1)| + (n-1) * |s(n-1, k)|$$

第二类: n 个元素的集定义 k 个等价类的方法数

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = S(n-1, k-1) + k * S(n-1, k)$$

三角公式

$$\sin(a \pm b) = \sin a \cos b \pm \cos a \sin b$$

$$\cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$$

$$\tan(a \pm b) = \frac{\tan(a) \pm \tan(b)}{1 \mp \tan(a) \tan(b)}$$

$$\tan(a) \pm \tan(b) = \frac{\sin(a \pm b)}{\cos(a) \cos(b)}$$

$$\sin(a) + \sin(b) = 2 \sin\left(\frac{a+b}{2}\right) \cos\left(\frac{a-b}{2}\right)$$

$$\sin(a) - \sin(b) = 2 \cos\left(\frac{a+b}{2}\right) \sin\left(\frac{a-b}{2}\right)$$

$$\cos(a) + \cos(b) = 2 \cos\left(\frac{a+b}{2}\right) \cos\left(\frac{a-b}{2}\right)$$

$$\cos(a) - \cos(b) = -2 \sin\left(\frac{a+b}{2}\right) \sin\left(\frac{a-b}{2}\right)$$

$$\sin(na) = n \cos^{n-1} a \sin a - \binom{n}{3} \cos^{n-3} a \sin^3 a + \binom{n}{5} \cos^{n-5} a \sin^5 a - \dots$$

$$\cos(na) = \cos^n a - \binom{n}{2} \cos^{n-2} a \sin^2 a + \binom{n}{4} \cos^{n-4} a \sin^4 a - \dots$$