New Meta

For Manual/Intelligence

August 6, 2018

Contents

1 Geometry 2
1.1 三维几何 2
1.2 三维凸包 2
1.3 阿波罗尼茨圆 2
1.4 最小覆盖球 2
1.5 三角形与圆交 3
1.6 圆并 3
1.7 Delaunay 三角剖分 3
1.8 二维几何 4
1.9 整数半平面交 5
1.10 凸包闵可夫斯基和 6
1.11 三角形 6
1.12 经纬度求球面最短距离 6
1.13 长方体表面两点最短距离 6
1.14 点到凸包切线 6
1.15 直线与凸包的交点 7

2 Graph 7
2.1 无向图最小割 7
2.2 Blossom Algorithm 7
2.3 仙人掌 8
2.4 最小树形图 8
2.5 Dominator Tree 8
2.6 离线动态最小生成树 9
2.7 GH Tree 9
2.8 Hopcroft mathcing 9
2.9 KM 10
2.10 Maximum Clique 10
2.11 原始对偶费用流 11
2.12 完美消除序列 11
2.13 Tarjan 12
2.14 ZKW 费用流 12

3 String 13
3.1 Exkmp 13
3.2 Lyndon Word Decomposition 13
3.3 Manacher 13
3.4 Minimum Representation 13
3.5 Palindromic Automaton 13
3.6 Suffix Array 13
3.7 Suffix Automaton 13
3.8 AC 自动机 14
3.9 子串最长公共子序列 14

4 Tree 14
4.1 树点分治-斜率优化 14
4.2 树链剖分 14
4.3 虚树 14
4.4 有根树同构 15

5 Math 15
5.1 Conclusions 15
5.2 积性函数线性求法 15
5.3 平方剩余 15
5.4 线性同余不等式 15
5.5 Schreier Sims 16
5.6 CRT 16
5.7 Factorial Mod 17
5.8 Miller Rabin and Pollard Rho 17
5.9 Pell 方程 17
5.10 Simplex 17
5.11 Simpson 18
5.12 FFT 18
5.13 解一元三次方程 19
5.14 线性递推 19
5.15 黑盒子代数 19

6 Data Structure 20
6.1 可持久化左偏树-K短路 20
6.2 左偏树 20
6.3 KD 树 20
6.4 LCT 21
6.5 Merge-Split Treap 21
6.6 Splay 21

7 Miscellany 22
7.1 日期公式 22
7.2 DLX - 主代码手 22
7.3 直线下格点统计 22

8 Others 23
8.1 Java Template 23
8.2 Formulas 23
8.2.1 Arithmetic Function 23
8.2.2 Binomial Coefficients 23
8.2.3 Fibonacci Numbers 24
8.2.4 Stirling Cycle Numbers 24
8.2.5 Stirling Subset Numbers 24
8.2.6 Eulerian Numbers 24
8.2.7 Harmonic Numbers 24
8.2.8 Pentagonal Number Theorem 24
8.2.9 Bell Numbers 24
8.2.10 Bernoulli Numbers 24
8.2.11 Tetrahedron Volume 24
8.2.12 BEST Thoerem 24
8.2.13 重心 24
8.2.14 Others 24
8.3 Integration Table 25
8.3.1 $ax^2+bx+c(a>0)$ 25
8.3.2 $\sqrt{\pm ax^2+bx+c}(a>0)$ 25
8.3.3 $\sqrt{\pm\frac{x-a}{x-b}}$或$\sqrt{(x-a)(x-b)}$ 25
8.3.4 三角函数的积分 25
8.3.5 反三角函数的积分(其中 $a>0$) 25
8.3.6 指数函数的积分 25
8.3.7 对数函数的积分 25

1. Geometry

1.1 三维几何

```
/* 大拇指指向x轴正方向时，4指弯曲由y轴正方向指向z轴正方向
   大拇指沿着原点到点(x， y， z)的向量，4指弯曲方向旋转w度 */
/* (x, y, z) * A = (x_new, y_new, z_new), 行向量右乘转移矩阵 */
void calc(D x, D y, D z, D w) {
  w = w * pi / 180;
  memset(a, 0, sizeof(a));
  s1 = x * x + y * y + z * z;
  a[0][0] = ((y*y+z*z)*cos(w)+x*x)/s1; a[0][1] = x*y*(1-cos(w))/s1+z*sin(w)/sqrt(s1); a[0][2] = x*z*(1-cos(w))/s1-y*sin(w)/sqrt(s1);
  a[1][0] = x*y*(1-cos(w))/s1-z*sin(w)/sqrt(s1); a[1][1] = ((x*x+z*z)*cos(w)+y*y)/s1; a[1][2] = y*z*(1-cos(w))/s1+x*sin(w)/sqrt(s1);
  a[2][0] = x*z*(1-cos(w))/s1+y*sin(w)/sqrt(s1); a[2][1] = y*z*(1-cos(w))/s1-x*sin(w)/sqrt(s1); a[2][2] = ((x*x+y*y)*cos(w)+z*z)/s1;
}
// 求平面和直线的交点
Point3D intersection(const Point3D &a, const Point3D &b, const Point3D &c, const Point3D &l0, const Point3D &l1) {
  Point3D p = pVec(a, b, c); // 平面法向量
  double t = (p.x * (a.x - l0.x) + p.y * (a.y - l0.y) + p.z * (a.z - l0.z)) / (p.x * (l1.x - l0.x) + p.y * (l1.y - l0.y) + p.z * (l1.z - l0.z));
  return l0 + (l1 - l0) * t;
}
```

1.2 三维凸包

```
int mark[N][N], cnt;
D mix(const Point & a, const Point & b, const Point & c) { return a.dot(b.cross(c)); }
double volume(int a, int b, int c, int d) { return mix(info[b] - info[a], info[c] - info[a], info[d] - info[a]); }
typedef array<int, 3> Face; vector<Face> face;
inline void insert(int a, int b, int c) { face.push_back({a, b, c}); }
void add(int v) {
  vector<Face> tmp; int a, b, c; cnt++;
  for(auto f : face)
    if(sign(volume(v, f[0], f[1], f[2])) < 0)
      for(int i : f) for(int j : f) mark[i][j] = cnt;
    else tmp.push_back(f);
  face = tmp;
  for(int i(0); i < (int)tmp.size(); i++) {
    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if(mark[a][b] == cnt) insert(b, a, v);
    if(mark[b][c] == cnt) insert(c, b, v);
    if(mark[c][a] == cnt) insert(a, c, v);
  }
}
int Find(int n) {
  for(int i(2); i < n; i++) {
    Point ndir = (info[0] - info[i]).cross(info[1] - info[i]);
    if(ndir == Point(0, 0, 0)) continue; swap(info[i], info[2]);
    for(int j = i + 1; j < n; j++) if(sign(volume(0, 1, 2, j)) != 0) {
      swap(info[j], info[3]); insert(0, 1, 2), insert(0, 2, 1); return 1;
    }
  }
}
int main() {
  int n; scanf("%d", &n);
  for(int i(0); i < n; i++) info[i].scan();
  random_shuffle(info, info + n);
  Find(n);
  for(int i = 3; i < n; i++) add(i);
}
```

1.3 阿波罗尼茨圆

```
硬币问题：易知两两相切的圆半径为 r1, r2, r3, 求与他们都相切的圆的半径 r4
分母取负号，答案再取绝对值，为外切圆半径
分母取正号为内切圆半径
// $r_4^{\pm} = \frac{r_1 r_2 r_3}{r_1 r_2 + r_1 r_3 + r_2 r_3 \pm 2\sqrt{r_1 r_2 r_3 (r_1 + r_2 + r_3)}}$
```

1.4 最小覆盖球

```
// 注意，无法处理小于四点的退化情况
struct P;
P a[33];
P intersect(const Plane & a, const Plane & b, const Plane & c) {
  P c1(a.nor.x, b.nor.x, c.nor.x), c2(a.nor.y, b.nor.y, c.nor.y), c3(a.nor.z, b.nor.z, c.nor.z), c4(a.m, b.m, c.m);
  return 1 / ((c1 * c2) % c3) * Point((c4 * c2) % c3, (c1 * c4) % c3, (c1 * c2) % c4);
}
bool in(const P & a, const Circle & b) {
  return sign((a - b.o).len() - b.r) <= 0;
}
vector<P> vec;
Circle calc() {
  if (vec.empty()) {
    return Circle(Point(0, 0, 0), 0);
  } else if(1 == (int)vec.size()) {
    return Circle(vec[0], 0);
  } else if(2 == (int)vec.size()) {
    return Circle(0.5 * (vec[0] + vec[1]), 0.5 * (vec[0] - vec[1]).len());
  } else if(3 == (int)vec.size()) {
    double r((vec[0] - vec[1]).len() * (vec[1] - vec[2]).len() * (vec[2] - vec[0]).len() / 2 /
      fabs(((vec[0] - vec[2]) * (vec[1] - vec[2])).len()));
    return Circle(intersect(Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0])),
          Plane(vec[2] - vec[1], 0.5 * (vec[2] + vec[1])),
        Plane((vec[1] - vec[0]) * (vec[2] - vec[0]), vec[0])), r);
  } else {
    P o(intersect(Plane(vec[1] - vec[0], 0.5 * (vec[1] + vec[0])),
        Plane(vec[2] - vec[0], 0.5 * (vec[2] + vec[0])),
        Plane(vec[3] - vec[0], 0.5 * (vec[3] + vec[0]))));
    return Circle(o, (o - vec[0]).len());
  }
}
Circle miniBall(int n) {
  Circle res(calc());
  for(int i(0); i < n; i++) {
    if(!in(a[i], res)) {
      vec.push_back(a[i]);
      res = miniBall(i);
      vec.pop_back();
      if (i) { Point tmp(a[i]); memmove(a + 1, a, sizeof(Point) * i); a[0] = tmp; }
    }
  }
  return res;
}
```

```cpp
int main() {
  for(int i(0); i < n; i++) a[i].scan();
  sort(a, a + n);
  n = unique(a, a + n) - a;
  vec.clear();
  random_shuffle(a, a + n);
  printf("%.10f\n", miniBall(n).r);
}
```

1.5 三角形与圆交

```cpp
// 反三角函数要在 [-1, 1] 中，sqrt 要与 0 取 max 别忘了取正负
// 改成周长请用注释，res1 为直线长度，res2 为弧线长度
// 多边形与圆求交时，相切精度比较差
D areaCT(P pa, P pb, D r) { //, D & res1, D & res2) {
    if (pa.len() < pb.len()) swap(pa, pb);
    if (sign(pb.len()) == 0) return 0; // if (sign(pb.len()) == 0) { res1 += min(r, pa.len()); return; }
    D a = pb.len(), b = pa.len(), c = (pb - pa).len();
    D sinB = fabs(pb * (pb - pa)), cosB = pb % (pb - pa), area = fabs(pa * pb);
    D S, B = atan2(sinB, cosB), C = atan2(area, pa % pb);
    sinB /= a * c; cosB /= a * c;
    if (a > r) {
        S = C / 2 * r * r; D h = area / c;//res2 += -1 * sgn * C * r; D h = area / c;
        if (h < r && B < pi / 2) {
            //res2 -= -1 * sgn * 2 * acos(max((D)-1., min((D)1., h / r))) * r;
            //res1 += 2 * sqrt(max((D)0., r * r - h * h));
            S -= (acos(max((D)-1., min((D)1., h / r))) * r * r - h * sqrt(max((D)0. ,r * r - h * h)));
        }
    } else if (b > r) {
        D theta = pi - B - asin(max((D)-1., min((D)1., sinB / r * a)));
        S = a * r * sin(theta) / 2 + (C - theta) / 2 * r * r;
        //res2 += -1 * sgn * (C - theta) * r;
        //res1 += sqrt(max((D)0., r * r + a * a - 2 * r * a * cos(theta)));
    } else S = area / 2; //res1 += (pb - pa).len();
    return S;
}
```

1.6 圆并

```cpp
struct Event {
  P p; D ang; int delta;
  Event (P p = Point(0, 0), D ang = 0, int delta = 0) : p(p), ang(ang), delta(delta) {}
};
bool operator < (const Event &a, const Event &b) { return a.ang < b.ang; }
void addEvent(const Circle &a, const Circle &b, vector<Event> &evt, int &cnt) {
  D d2 = (a.o - b.o).sqrlen(), dRatio = ((a.r - b.r) * (a.r + b.r) / d2 + 1) / 2,
    pRatio = sqrt(max((D)0., -(d2 - sqr(a.r - b.r)) * (d2 - sqr(a.r + b.r)) / (d2 * d2 * 4)));
  P d = b.o - a.o, p = d.rot(pi / 2),
    q0 = a.o + d * dRatio + p * pRatio,
    q1 = a.o + d * dRatio - p * pRatio;
  D ang0 = (q0 - a.o).ang(), ang1 = (q1 - a.o).ang();
  evt.emplace_back(q1, ang1, 1); evt.emplace_back(q0, ang0, -1);
  cnt += ang1 > ang0;
}
bool issame(const Circle &a, const Circle &b) { return sign((a.o - b.o).len()) == 0 && sign(a.r - b.r) == 0; }
bool overlap(const Circle &a, const Circle &b) { return sign(a.r - b.r - (a.o - b.o).len()) >= 0; }
bool intersect(const Circle &a, const Circle &b) { return sign((a.o - b.o).len() - a.r - b.r) < 0; }
int C;
Circle c[N];
double area[N];
void solve() { // 返回覆盖至少 k 次的面积
  memset(area, 0, sizeof(D) * (C + 1));
  for (int i = 0; i < C; ++i) {
    int cnt = 1;
    vector<Event> evt;
    for (int j = 0; j < i; ++j) if (issame(c[i], c[j])) ++cnt;
    for (int j = 0; j < C; ++j)
      if (j != i && !issame(c[i], c[j]) && overlap(c[j], c[i]))
        ++cnt;
    for (int j = 0; j < C; ++j)
      if (j != i && !overlap(c[j], c[i]) && !overlap(c[i], c[j]) && intersect(c[i], c[j]))
        addEvent(c[i], c[j], evt, cnt);
    if (evt.empty()) area[cnt] += PI * c[i].r * c[i].r;
    else {
      sort(evt.begin(), evt.end());
      evt.push_back(evt.front());
      for (int j = 0; j + 1 < (int)evt.size(); ++j) {
        cnt += evt[j].delta;
        area[cnt] += det(evt[j].p, evt[j + 1].p) / 2;
        D ang = evt[j + 1].ang - evt[j].ang;
        if (ang < 0) ang += PI * 2;
        area[cnt] += ang * c[i].r * c[i].r / 2 - sin(ang) * c[i].r * c[i].r / 2;
} } } }
```

1.7 Delaunay 三角剖分

```cpp
/*
Delaunay Triangulation 随机增量算法 :
节点数至少为点数的 6 倍，空间消耗较大注意计算内存使用
建图的过程在 build 中，注意初始化内存池和初始三角形的坐标范围 (Triangulation::LOTS)
Triangulation::find 返回包含某点的三角形
Triangulation::add_point 将某点加入三角剖分
某个 Triangle 在三角剖分中当且仅当它的 has_children 为 0
如果要找到三角形 u 的邻域，则枚举它的所有 u.edge[i].tri, 该条边的两个点为 u.p[(i+1)%3], u.p[(i+2)%3]
*/
const int N = 100000 + 5, MAX_TRIS = N * 6;
const double EPSILON = 1e-6, PI = acos(-1.0);
struct Point {
  double x,y; Point():x(0),y(0){}
    Point(double x, double y):x(x),y(y){}
  bool operator ==(Point const& that)const {return x==that.x&&y==that.y;}
};
inline double sqr(double x) { return x*x; }
double dist_sqr(Point const& a, Point const& b){return sqr(a.x-b.x)+sqr(a.y-b.y);}
bool in_circumcircle(Point const& p1, Point const& p2, Point const& p3, Point const& p4) {
  double u11 = p1.x - p4.x, u21 = p2.x - p4.x, u31 = p3.x - p4.x;
  double u12 = p1.y - p4.y, u22 = p2.y - p4.y, u32 = p3.y - p4.y;
  double u13 = sqr(p1.x) - sqr(p4.x) + sqr(p1.y) - sqr(p4.y);
  double u23 = sqr(p2.x) - sqr(p4.x) + sqr(p2.y) - sqr(p4.y);
```

```
    double u33 = sqr(p3.x) - sqr(p4.x) + sqr(p3.y) - sqr(p4.y);
    double det = -u13*u22*u31 + u12*u23*u31 + u13*u21*u32 - u11*u23*u32 - u12*u21*u33 + u11*u22*u33;
    return det > EPSILON;
}
double side(Point const& a, Point const& b, Point const& p) { return (b.x-a.x)*(p.y-a.y) - (b.y-a.y)*(p.x-a.x);}
typedef int SideRef; struct Triangle; typedef Triangle* TriangleRef;
struct Edge {
    TriangleRef tri; SideRef side; Edge() : tri(0), side(0) {}
    Edge(TriangleRef tri, SideRef side) : tri(tri), side(side) {}
};
struct Triangle {
    Point p[3]; Edge edge[3]; TriangleRef children[3]; Triangle() {}
    Triangle(Point const& p0, Point const& p1, Point const& p2) {
      p[0] = p0; p[1] = p1; p[2] = p2;
          children[0] = children[1] = children[2] = 0;
    }
    bool has_children() const { return children[0] != 0; }
    int num_children() const {
      return children[0] == 0 ? 0
        : children[1] == 0 ? 1
        : children[2] == 0 ? 2 : 3;
    }
    bool contains(Point const& q) const {
      double a=side(p[0],p[1],q), b=side(p[1],p[2],q), c=side(p[2],p[0],q);
      return a >= -EPSILON && b >= -EPSILON && c >= -EPSILON;
    }
} triange_pool[MAX_TRIS], *tot_triangles;
void set_edge(Edge a, Edge b) {
    if (a.tri) a.tri->edge[a.side] = b;
    if (b.tri) b.tri->edge[b.side] = a;
}
class Triangulation {
    public:
      Triangulation() {
        const double LOTS = 1e6;
        the_root = new(tot_triangles++) Triangle(Point(-LOTS,-LOTS),Point(+LOTS,-LOTS),Point(0,+LOTS));
      }
      TriangleRef find(Point p) const { return find(the_root,p); }
      void add_point(Point const& p) { add_point(find(the_root,p),p); }
    private:
      TriangleRef the_root;
      static TriangleRef find(TriangleRef root, Point const& p) {
        for( ; ; ) {
          if (!root->has_children()) return root;
          else for (int i = 0; i < 3 && root->children[i] ; ++i)
              if (root->children[i]->contains(p))
                {root = root->children[i]; break;}
        }
      }
      void add_point(TriangleRef root, Point const& p) {
        TriangleRef tab,tbc,tca;
        tab = new(tot_triangles++) Triangle(root->p[0], root->p[1], p);
        tbc = new(tot_triangles++) Triangle(root->p[1], root->p[2], p);
        tca = new(tot_triangles++) Triangle(root->p[2], root->p[0], p);
        set_edge(Edge(tab,0),Edge(tbc,1)); set_edge(Edge(tbc,0),Edge(tca,1));
        set_edge(Edge(tca,0),Edge(tab,1)); set_edge(Edge(tab,2),root->edge[2]);
        set_edge(Edge(tbc,2),root->edge[0]); set_edge(Edge(tca,2),root->edge[1]);
        root->children[0]=tab; root->children[1]=tbc; root->children[2]=tca;
        flip(tab,2); flip(tbc,2); flip(tca,2);
      }
      void flip(TriangleRef tri, SideRef pi) {
        TriangleRef trj = tri->edge[pi].tri; int pj = tri->edge[pi].side;
        if(!trj || !in_circumcircle(tri->p[0],tri->p[1],tri->p[2],trj->p[pj])) return;
        TriangleRef trk = new(tot_triangles++) Triangle(tri->p[(pi+1)%3], trj->p[pj], tri->p[pi]);
        TriangleRef trl = new(tot_triangles++) Triangle(trj->p[(pj+1)%3], tri->p[pi], trj->p[pj]);
        set_edge(Edge(trk,0), Edge(trl,0));
        set_edge(Edge(trk,1), tri->edge[(pi+2)%3]); set_edge(Edge(trk,2), trj->edge[(pj+1)%3]);
        set_edge(Edge(trl,1), trj->edge[(pj+2)%3]); set_edge(Edge(trl,2), tri->edge[(pi+1)%3]);
        tri->children[0]=trk; tri->children[1]=trl; tri->children[2]=0;
        trj->children[0]=trk; trj->children[1]=trl; trj->children[2]=0;
        flip(trk,1); flip(trk,2); flip(trl,1); flip(trl,2);
      }
};
int n; Point ps[N];
void build(){
    tot_triangles = triange_pool; cin >> n;
    for(int i = 0; i < n; ++ i) scanf("%lf%lf",&ps[i].x,&ps[i].y);
    random_shuffle(ps, ps + n); Triangulation tri;
    for(int i = 0; i < n; ++ i) tri.add_point(ps[i]);
}
```

1.8 二维几何

```
// 求圆与直线的交点
bool isCL(Circle a, Line l, P &p1, P &p2) {
    D x = (l.s - a.o) % l.d,
      y = l.d.sqrlen(),
      d = x * x - y * ((l.s - a.o).sqrlen() - a.r * a.r);
    if (sign(d) < 0) return false;
    P p = l.s - x / y * l.d, delta = sqrt(max((D)0., d)) / y * l.d;
    p1 = p + delta, p2 = p - delta;
    return true;
}
// 求圆与圆的交面积
D areaCC(const Circle &c1, const Circle &c2) {
    D d = (c1.o - c2.o).len();
    if (sign(d - (c1.r + c2.r)) >= 0) {
      return 0;
    }
    if (sign(d - abs(c1.r - c2.r)) <= 0) {
      D r = min(c1.r, c2.r);
      return r * r * pi;
    }
    D x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d),
         t1 = acos(min(1., max(-1., x / c1.r))), t2 = acos(min(1., max(-1., (d - x) / c2.r)));
    return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r * sin(t1);
}
```

```cpp
// 求圆与圆的交点，注意调用前要先判定重圆
bool isCC(Circle a, Circle b, P &p1, P &p2) {
  D s1 = (a.o - b.o).len();
  if (sign(s1 - a.r - b.r) > 0 || sign(s1 - abs(a.r - b.r)) < 0) return false;
  D s2 = (a.r * a.r - b.r * b.r) / s1;
  D aa = (s1 + s2) * 0.5, bb = (s1 - s2) * 0.5;
  P o = aa / (aa + bb) * (b.o - a.o) + a.o;
  P delta = sqrt(max(0., a.r * a.r - aa * aa)) * (b.o - a.o).zoom(1).rev();
  p1 = o + delta, p2 = o - delta;
  return true;
}
// 求点到圆的切点，按关于点的顺时针方向返回两个点，rev 必须是 (-y, x)
bool tanCP(const Circle &c, const P &p0, P &p1, P &p2) {
  D x = (p0 - c.o).sqrlen(), d = x - c.r * c.r;
  if (d < eps) return false; // 点在圆上认为没有切点
  P p = c.r * c.r / x * (p0 - c.o);
  P delta = (-c.r * sqrt(d) / x * (p0 - c.o)).rev();
  p1 = c.o + p + delta;
  p2 = c.o + p - delta;
  return true;
}
// 求圆到圆的外共切线，按关于 c1.o 的顺时针方向返回两条线，rev 必须是 (-y, x)
vector<Line> extanCC(const Circle &c1, const Circle &c2) {
  vector<Line> ret;
  if (sign(c1.r - c2.r) == 0) {
    P dir = c2.o - c1.o;
    dir = (c1.r / dir.len() * dir).rev();
    ret.push_back(Line(c1.o + dir, c2.o - c1.o));
    ret.push_back(Line(c1.o - dir, c2.o - c1.o));
  } else {
    P p = 1. / (c1.r - c2.r) * (-c2.r * c1.o + c1.r * c2.o);
    P p1, p2, q1, q2;
    if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) {
      if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
      ret.push_back(Line(p1, q1 - p1));
      ret.push_back(Line(p2, q2 - p2));
    }
  }
  return ret;
}
// 求圆到圆的内共切线，按关于 c1.o 的顺时针方向返回两条线，rev 必须是 (-y, x)
vector<Line> intanCC(const Circle &c1, const Circle &c2) {
  vector<Line> ret;
  P p = 1. / (c1.r + c2.r) * (c2.r * c1.o + c1.r * c2.o);
  P p1, p2, q1, q2;
  if (tanCP(c1, p, p1, p2) && tanCP(c2, p, q1, q2)) { // 两圆相切认为没有切线
    ret.push_back(Line(p1, q1 - p1));
    ret.push_back(Line(p2, q2 - p2));
  }
  return ret;
}
bool contain(vector<P> poly, P p) { // 判断点 p 是否被多边形包含，包括落在边界上
  int ret = 0, n = poly.size();
  for(int i = 0; i < n; ++ i) {
    P u = poly[i], v = poly[(i + 1) % n];
    if (onSeg(p, u, v)) return true; // 在边界上
    if (sign(u.y - v.y) <= 0) swap(u, v);
    if (sign(p.y - u.y) > 0 || sign(p.y - v.y) <= 0) continue;
    ret += sign((v - p) * (u - p)) > 0;
  }
  return ret & 1;
}
vector<P> convexCut(const vector<P>&ps, Line l) { // 用半平面 (s,d) 的逆时针方向去切凸多边形
  vector<P> qs;
  int n = ps.size();
  for (int i = 0; i < n; ++i) {
    Point p1 = ps[i], p2 = ps[(i + 1) % n];
    int d1 = sign(l.d * (p1 - l.s)), d2 = sign(l.d * (p2 - l.s));
    if (d1 >= 0) qs.push_back(p1);
    if (d1 * d2 < 0) qs.push_back(isLL(Line(p1, p2 - p1), l));
  }
  return qs;
}
```

1.9 整数半平面交

```cpp
typedef __int128 J; // 坐标 |1e9| 就要用 int128 来判断
struct Line {
  bool include(P a) const { return (a - s) * d >= 0; } // 严格去掉 =
  bool include(Line a, Line b) const {
    J l1(a.d * b.d);
    if(!l1)  return true;
    J x(l1 * (a.s.x - s.x)), y(l1 * (a.s.y - s.y));
    J l2((b.s - a.s) * b.d);
    x += l2 * a.d.x; y += l2 * a.d.y;
    J res(x * d.y - y * d.x);
    return l1 > 0 ? res >= 0 : res <= 0; // 严格去掉 =
  }
};
bool HPI(vector<Line> v) { // 返回 v 中每个射线的右侧的交是否非空
  sort(v.begin(), v.end());// 按方向排极角序
  { // 同方向取最严格的一个
    vector<Line> t; int n(v.size());
    for(int i(0), j; i < n; i = j) {
      LL mx(-9e18); int mxi;
      for(j = i; j < n && v[i].d * v[j].d == 0; j++) {
        LL tmp(v[j].s * v[i].d);
        if(tmp > mx)
          mx = tmp, mxi = j;
      }
      t.push_back(v[mxi]);
    }
    swap(v, t);
  }
  deque<Line> res;
  bool emp(false);
  for(auto i : v) {
    if(res.size() == 1) {
      if(res[0].d * i.d == 0 && !i.include(res[0].s))  {
        res.pop_back();
        emp = true;
      }
    } else if(res.size() >= 2) {
      while(res.size() >= 2u && !i.include(res.back(), res[res.size() - 2])) {
        if(i.d * res[res.size() - 2].d == 0 || !res.back().include(i, res[res.size() - 2])) {
          emp = true;
          break;
        }
        res.pop_back();
      }
      while(res.size() >= 2u && !i.include(res[0], res[1])) res.pop_front();
    }
    if(emp) break;
    res.push_back(i);
  }
  while (res.size() > 2u && !res[0].include(res.back(), res[res.size() - 2])) res.pop_back();
```

```
    return !emp;// emp: 是否为空，res 按顺序即为半平面交
}
```

1.10 凸包闵可夫斯基和

```
// cv[0..1] 为两个顺时针凸包，其中起点等于终点，求出的闵可夫斯基和不一定是严格凸包
int i[2] = {0, 0}, len[2] = {(int)cv[0].size() - 1, (int)cv[1].size() - 1};
vector<P> mnk;
mnk.push_back(cv[0][0] + cv[1][0]);
do {
  int d((cv[0][i[0] + 1] - cv[0][i[0]]) * (cv[1][i[1] + 1] - cv[1][i[1]]) >= 0);
  mnk.push_back(cv[d][i[d] + 1] - cv[d][i[d]] + mnk.back());
  i[d] = (i[d] + 1) % len[d];
} while(i[0] || i[1]);
```

1.11 三角形

```
P fermat(const P& a, const P& b, const P& c) {
  D ab((b - a).len()), bc((b - c).len()), ca((c - a).len());
  D cosa((b - a) % (c - a) / ab / ca);
  D cosb((a - b) % (c - b) / ab / bc);
  D cosc((b - c) % (a - c) / ca / bc);
  P mid; D sq3(sqrt(3) / 2);
  if(sign((b - a) * (c - a)) < 0) swap(b, c);
  if(sign(cosa + 0.5) < 0) mid = a;
  else if(sign(cosb + 0.5) < 0) mid = b;
  else if(sign(cosc + 0.5) < 0) mid = c;
  else mid = intersection(Line(a, c + (b - c).rot(sq3) - a), Line(c, b + (a - b).rot(sq3) - c));
  return mid;
  // mid 为三角形 abc 费马点，要求 abc 非退化
  length = (mid - a).len() + (mid - b).len() + (mid - c).len();
  // 以下求法仅在三角形三个角均小于120度时，可以求出ans为费马点到abc三点距离和
  length = (a - c - (b - c).rot(sq3)).len();
}
P inCenter(const P & A, const P & B, const P & C) { // 内心
  D a = (B - C).len(), b = (C - A).len(), c = (A - B).len(),
    s = abs((B - A) * (C - A)),
    r = s / (a + b + c); // 内接圆半径
  return 1. / (a + b + c) * (A * a + B * b + C * c); // 偏心则将对应点前两个加号改为减号
}
P circumCenter(const P & a, const P & b, const P & c) { // 外心
  P bb = b - a, cc = c - a;
  // 半径为 a * b * c / 4 / S, a, b, c 为边长，S 为面积
  D db = bb.sqrlen(), dc = cc.sqrlen(), d = 2 * (bb * cc);
  return a - 1. / d * P(bb.y * dc - cc.y * db, cc.x * db - bb.x * dc);
}
P othroCenter(const P & a, const P & b, const P & c) { // 垂心
  P ba = b - a, ca = c - a, bc = b - c;
  D Y = ba.y * ca.y * bc.y,
      A = ca.x * ba.y - ba.x * ca.y,
      x0 = (Y + ca.x * ba.y * b.x - ba.x * ca.y * c.x) / A,
      y0 = -ba.x * (x0 - c.x) / ba.y + ca.y;
  return P(x0, y0);
}
```

1.12 经纬度求球面最短距离

```
double sphereDis(double lon1, double lat1, double lon2, double lat2, double R) {
  return R * acos(cos(lat1) * cos(lat2) * cos(lon1 - lon2) + sin(lat1) * sin(lat2));
}
```

1.13 长方体表面两点最短距离

```
int r;
void turn(int i, int j, int x, int y, int z,int x0, int y0, int L, int W, int H) {
  if (z==0) { int R = x*x+y*y; if (R<r) r=R;
  } else {
    if(i>=0 && i< 2) turn(i+1, j, x0+L+z, y, x0+L-x, x0+L, y0, H, W, L);
    if(j>=0 && j< 2) turn(i, j+1, x, y0+W+z, y0+W-y, x0, y0+W, L, H, W);
    if(i<=0 && i>-2) turn(i-1, j, x0-z, y, x-x0, x0-H, y0, H, W, L);
    if(j<=0 && j>-2) turn(i, j-1, x, y0-z, y-y0, x0, y0-H, L, H, W);
  }
}
int main(){
  int L, H, W, x1, y1, z1, x2, y2, z2;
  cin >> L >> W >> H >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
  if (z1!=0 && z1!=H) if (y1==0 || y1==W)
       swap(y1,z1), std::swap(y2,z2), std::swap(W,H);
  else swap(x1,z1), std::swap(x2,z2), std::swap(L,H);
  if (z1==H) z1=0, z2=H-z2;
  r=0x3fffffff;
  turn(0,0,x2-x1,y2-y1,z2,-x1,-y1,L,W,H);
  cout<<r<<endl;
}
```

1.14 点到凸包切线

```
P lb(P x, vector<P> & v, int le, int ri, int sg) {
    if (le > ri) le = ri;
    int s(le), t(ri);
    while (le != ri) {
        int mid((le + ri) / 2);
        if (sign((v[mid] - x) * (v[mid + 1] - v[mid])) == sg)
            le = mid + 1; else ri = mid;
    }
    return x - v[le]; // le 即为下标，按需返回
}
// v[0] 为顺时针上凸壳，v[1] 为顺时针下凸壳，均允许起始两个点横坐标相同
// 返回值为真代表严格在凸包外，顺时针旋转在 d1 方向先碰到凸包
bool getTan(P x, vector<P> * v, P & d1, P & d2) {
    if (x.x < v[0][0].x) {
        d1 = lb(x, v[0], 0, sz(v[0]) - 1, 1);
        d2 = lb(x, v[1], 0, sz(v[1]) - 1, -1);
        return true;
    } else if(x.x > v[0].back().x) {
        d1 = lb(x, v[1], 0, sz(v[1]) - 1, 1);
        d2 = lb(x, v[0], 0, sz(v[0]) - 1, -1);
        return true;
    } else {
        for(int d(0); d < 2; d++) {
            int id(lower_bound(v[d].begin(), v[d].end(), x,
            [&](const P & a, const P & b) {
                return d == 0 ? a < b : b < a;
            }) - v[d].begin());
            if (id && (id == sz(v[d]) || (v[d][id - 1] - x) * (v[d][id] - x) > 0)) {
                d1 = lb(x, v[d], id, sz(v[d]) - 1, 1);
```

```
                d2 = lb(x, v[d], 0, id, -1);
                return true;
            }
        }
    }
    return false;
}
```

1.15 直线与凸包的交点

```
// a 是顺时针凸包，i1 为 x 最小的点，j1 为 x 最大的点 需保证 j1 > i1
// n 是凸包上的点数，a 需复制多份或写循环数组类
int lowerBound(int le, int ri, const P & dir) {
  while (le < ri) {
    int mid((le + ri) / 2);
    if (sign((a[mid + 1] - a[mid]) * dir) <= 0) {
      le = mid + 1;
    } else ri = mid;
  }
  return le;
}
int boundLower(int le, int ri, const P & s, const P & t) {
  while (le < ri) {
    int mid((le + ri + 1) / 2);
    if (sign((a[mid] - s) * (t - s)) <= 0) {
      le = mid;
    } else ri = mid - 1;
  }
  return le;
}

void calc(P s, P t) {
  if(t < s) swap(t, s);
  int i3(lowerBound(i1, j1, t - s)); // 和上凸包的切点
  int j3(lowerBound(j1, i1 + n, s - t)); // 和下凸包的切点
  int i4(boundLower(i3, j3, s, t)); // 如果有交则是右侧的交点，与 a[i4]~a[i4+1] 相交 要判断是否有交的话 就手动 check 一下
  int j4(boundLower(j3, i3 + n, t, s)); // 如果有交左侧的交点，与 a[j4]~a[j4+1] 相交
    // 返回的下标不一定在 [0 ~ n-1] 内
}
```

2. Graph

2.1 无向图最小割

```
//inf 比所有的值的和还要大
int cost[maxn][maxn], seq[maxn], len[maxn], n, m, pop, ans;
bool used[maxn];
void Init() {
  int i, j, a, b, c;
  for (i = 0; i < n; i++) for(j = 0; j < n; j++) cost[i][j] = 0;
  for (i = 0; i < m; i++) {
        // cin >> u >> v >> cost;
        // cost[u][v] += c; cost[v][u] += c;
    }
  pop = n; for(i = 0; i < n; i++) seq[i] = i;
}
void Work(){
  ans = inf; int i, j, k, l, mm, sum, pk;
  while (pop > 1){
    for(i = 1; i < pop; i++) used[seq[i]] = 0;
        used[seq[0]] = 1;
        for(i = 1; i < pop; i++) {
            len[seq[i]] = cost[seq[0]][seq[i]];
        } pk = 0; mm = -inf; k = -1;
    for(i = 1; i < pop; i++) if (len[seq[i]] > mm) {
            mm = len[seq[i]]; k = i;
        }
        for(i = 1; i < pop; i++) {
      used[seq[l = k]] = 1;
      if (i == pop - 2) pk = k;
      if (i == pop - 1) break;
      mm = -inf;
      for (j = 1; j < pop; j++) if(!used[seq[j]])
        if ((len[seq[j]] += cost[seq[l]][seq[j]]) > mm)
          mm = len[seq[j]], k = j;
    }
    sum = 0;
    for(i = 0; i < pop; i++) if(i != k) sum += cost[seq[k]][seq[i]];
    ans = min(ans, sum);
    for(i = 0; i < pop;i++)
      cost[seq[k]][seq[i]] = cost[seq[i]][seq[k]] += cost[seq[pk]][seq[i]];
    seq[pk] = seq[--pop];
  }
  printf("%d\n",ans);
}
```

2.2 Blossom Algorithm

```
// 0 base, O(V^3)
vector<int> adj[N], q;
int n, mat[N], pred[N], base[N], type[N];
int lca(int u, int v) {
  static int visit[N], tick = 0; ++tick;
  for (int i = 0; i < 2; i++, swap(u, v)) {
    for (u = base[u]; ~mat[u]; u = base[pred[mat[u]]]) {
      if (visit[u] == tick) return u;
      visit[u] = tick;
  }  } return u;
}
void contract(int u, int v, int o) {
  for (; base[u] != o; v = mat[u], u = pred[v]) {
    pred[u] = v;
    base[u] = base[mat[u]] = o;
    if (type[mat[u]] == 1) {
      type[mat[u]] = 2;
      q.push_back(mat[u]);
  }  }
}
bool augment(int start) { // O(V^2)
  for(int i = 0; i < n; ++i)
    pred[i] = -1, base[i] = i, type[i] = 0;
  q.clear();
  type[start] = 2; q.push_back(start);
  for (int head = 0; head < q.size(); head++) {
    int u = q[head];
    for (auto v : adj[u]) {
      if (type[v] == 0) {
        if (mat[v] == -1) {
          for (int tmp; v >= 0; v = tmp, u = pred[v])
            tmp = mat[u], mat[v] = u, mat[u] = v;
          return true;
        }
        pred[v] = u;
        q.push_back(mat[v]);
        type[v] = 1, type[mat[v]] = 2;
      } else if (type[v] == 2 && base[u] != base[v]) {
        int o = lca(u, v);
        contract(u, v, o), contract(v, u, o);
  }  }  }
  return false;
}
int blossom() {
  int num = 0; fill(mat, mat + n, -1);
  for(int i = 0; i < n; ++i) if (mat[i] == -1) num += augment(i);
```

```cpp
  return num;
}
```

2.3 仙人掌

```cpp
int fa[N], ma[N], stmp[N], tim;
bool ins[N], vst[N];
vector<int> adj[N];// ma: 环上右侧的点, fa: 树上的父亲, 或环上左边的点
vector<vector<int> > cycles[N];
void dfs(int v) {
  ins[v] = true; vst[v] = true;
  for(int y : adj[v])
    if(!vst[y]) {
      fa[y] = v;
      dfs(y);
    }else if(ins[y] && y != fa[v]) {
      cycles[y].push_back(vector<int>(1, y));
      int x(v);
      ma[v] = y;
      while(x != y) {
        cycles[y].back().push_back(x);
        if(fa[x] != y)
          ma[fa[x]] = x;
        x = fa[x];
      }
    }
  tim++;
  for(auto & cyc : cycles[v]) for(int y : cyc) {
      stmp[y] = tim;
      if(y != v);// 此处是环上的点
  }
  for(int y : adj[v]) if(y != fa[v] && y != ma[v] && stmp[y] != tim);// 此处是树上的儿子
  ins[v] = false;
}
void sfd(int v) {
  for(auto & cyc : cycles[v]) for(int y : cyc);// 枚举环上的点
  for(auto & cyc : cycles[v]) for(int y : cyc) if(y != v) sfd(y);
  tim++;
  for(auto & cyc : cycles[v]) for(int y : cyc) if(y != v) stmp[y] = tim;
  int tt(tim);
  for(int y : adj[v]) if(y != fa[v] && y != ma[v] && stmp[y] != tt) sfd(y);
}
```

2.4 最小树形图

```cpp
vector<pair<VAL, int>> G[N], fv[N][N];
int n, m, parent[N];
// O(V^2), add(u, v, w) -> fv[v][u] = {w, v};
// O(ElogE) 只需要使用支持打标记的可并堆维护即可
// DSU 为并查集, 需要重载 [], 不求方案时 VAL e[][]; 即可
VAL chuliu(int s) {
  VAL ret = 0; static DSU v, c; // int rid = 0;
  v.clear(n), c.clear(n);
  for (int u = 0; u < n; ++u) G[u].clear();
  for (int u = 0; u < n; ++u) if (u != s) {
    int uu = u;
    for (;;) {
      int p = s;
      for (int it = 0; it < n; ++it) if (v[it] != uu)
        p = fv[uu][it] < fv[uu][p] ? it : p;
      if (fv[uu][p].first == INF) return INF;
      ret += fv[uu][p].first, parent[uu] = p;
      // if (p == s) root = fv[uu][p].second; // 实根
      for (int it = 0; it < n; it++) if (it != p && fv[uu][it].first != INF)
        fv[uu][it].first -= fv[uu][p].first;
      if (c[p] != c[u]) { c.merge(u, p); break; }
      // G[p].push_back({fv[uu][p].second, ++rid});
      for (int j = v[p]; j != v[u]; j = v[parent[j]]) {
        // G[parent[j]].push_back({fv[j][parent[j]].second, rid});
        for (int k = 0; k < n; ++k) fv[j][k] = min(fv[j][k], fv[uu][k]);
        uu = v[u] = j;
    }}}
  // ++rid;
  // for (int i = 0; i < n; ++i) if (i != s && v[i] == i) {
  //   G[parent[i]].push_back({fv[i][parent[i]].second, rid}); }
  return ret;
}

void makeSol(int s) { // 用堆优化Prim构造方案
  static int dist[N];
  fill(dist, dist + n, 2 * n + 1); parent[s] = -1;
  for (multiset<pair<int, int>> h = {{0, s}}; !h.empty();) {
    int u = h.begin()->second; h.erase(h.begin()); dist[u] = 0;
    for (auto e : G[u]) if (e.second < dist[e.first]) {
      int v = e.first;
      h.erase({dist[v], v});
      h.insert({dist[v] = e.second, v});
      parent[v] = u; }}}
```

2.5 Dominator Tree

```cpp
// 1 base, O(m)
int n;
Array dfn, id, pa, semi, idom, p, mn; vector<int> be[N], dom[N]; int cnt;
vector<int> e[N];
void dfs(int x) {
  dfn[x] = ++cnt; id[cnt] = x;
  for (auto i : e[x]) {
    if (!dfn[i]) { dfs(i); pa[dfn[i]] = dfn[x]; }
    be[dfn[i]].pb(dfn[x]);
}}
int get(int x) {
  if (p[x] != p[p[x]]) {
        if (semi[mn[x]] > semi[get(p[x])]) mn[x] = get(p[x]);
    p[x] = p[p[x]];
  }
  return mn[x];
}
void LT() {
  for (int i = cnt; i > 1; --i) {
    for (auto j : be[i]) semi[i] = min(semi[i], semi[get(j)]);
    dom[semi[i]].pb(i);
    int x = p[i] = pa[i];
    for (auto j : dom[x])
      idom[j] = (semi[get(j)] < x ? get(j) : x);
    dom[x] = {};
  }
  for (int i = 2; i <= cnt; ++i) {
    if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
    dom[id[idom[i]]].pb(id[i]); // dom is dominator tree's son list
}}
void build(int s) {
  for (int i = 1; i <= n; ++i) {
    dfn[i] = 0; dom[i] = be[i] = {};
    p[i] = mn[i] = semi[i] = i;
  }
```

```
  cnt = 0; dfs(s); LT();
}
```

2.6 离线动态最小生成树

```
// O((m + q) log q)
int n, m, q;
struct EdgeInfo {
  int u, v, w, l, r;
  EdgeInfo(int u, int v, int w, int l, int r) : u(u),
    ↪ v(v), w(w), l(l), r(r) {}
  EdgeInfo() {}
};
long long ans[N];
int find(int f[], int u) {
  return f[u] == u ? u : f[u] = find(f, f[u]);
}
bool join(int f[], int u, int v) {
  u = find(f, u), v = find(f, v);
  if (u == v) return false;
  return f[u] = v, true;
}
void dfs(int l, int r, int n, const vector<EdgeInfo>
  ↪ &list, long long base) {
  if (list.empty()) {
    for (int i = l; i <= r; i++) ans[l] = base;
    return ;
  }
  static vector<EdgeInfo> all, part;
  all.clear();
  part.clear();
  for (auto &e : list) {
    if (e.l <= l && r <= e.r) {
      all.push_back(e);
    } else if (l <= e.r && e.l <= r) {
      part.push_back(e);
    }
  }
  static int f[N], color[N], id[N];
  // Contraction
  for (int i = 0; i < n; i++) f[i] = color[i] = i;
  for (auto &e : part) join(f, e.u, e.v);
  for (auto &e : all) if (join(f, e.u, e.v)) {
    join(color, e.u, e.v);
    base += e.w;
  }
  if (l == r) {
    ans[l] = base;
    return ;
  }
  for (int i = 0; i < n; i++) id[i] = -1;
  int tot = 0;
  for (int u = 0; u < n; u++) {
    int v = find(color, u);
    if (id[v] == -1) id[v] = tot++;
    id[u] = id[v];
  }
  // Reduction
  int m = 0;
  for (int i = 0; i < tot; i++) f[i] = i;
  for (auto &e : part) {
    e.u = id[find(color, e.u)];
    e.v = id[find(color, e.v)];
  }
  for (auto &e : all) {
    e.u = id[find(color, e.u)], e.v = id[find(color,
      ↪ e.v)];
    if (e.u == e.v) continue;
    assert(e.u < tot && e.v < tot);
    if (join(f, e.u, e.v)) all[m++] = e;
  }
  all.resize(m);
  vector<EdgeInfo> new_list;
  for (int i = 0, j = 0; i < part.size() || j <
    ↪ all.size(); ) {
    if (i < part.size() && (j == all.size() || all[j].w >
      ↪ part[i].w)) {
      new_list.push_back(part[i++]);
    } else {
      new_list.push_back(all[j++]);
    }
  }
  int mid = (l + r) / 2;
  dfs(l, mid, tot, new_list, base);
  dfs(mid + 1, r, tot, new_list, base);
}
int main() {
  scanf("%d %d %d", &n, &m, &q);
  vector<pair<int, int> > memo;
  static int u[N], v[N], w[N];
  for (int i = 0; i < m; i++) {
    scanf("%d %d %d", &u[i], &v[i], &w[i]);
    --u[i], --v[i];
    memo.push_back({0, w[i]});
  }
  vector<EdgeInfo> info;
   // 把第 k 条边权值改为 d
  for (int i = 0; i < q; i++) {
    int k, d; scanf("%d %d", &k, &d); --k;
    if (memo[k].first < i) {
      info.push_back({u[k], v[k], memo[k].second,
        ↪ memo[k].first, i - 1});
    } memo[k] = {i, d};
  }
  for (int i = 0; i < m; i++) {
    info.push_back({u[i], v[i], memo[i].second,
      ↪ memo[i].first, q - 1});
  }
  sort(info.begin(), info.end(), [&](const EdgeInfo &a,
    ↪ const EdgeInfo &b) { return a.w < b.w; });
  dfs(0, q - 1, n, info, 0);
  for (int i = 0; i < q; i++) {
    printf("%lld\n", ans[i]);
  }
  return 0;
}
```

2.7 GH Tree

```
void build(int *l, int *r) { // 左闭右开
  auto t = r - 1; if (l >= t) return;
  random_shuffle(l, r);
  G.reset(); // 重置流量
  add2(*l, *t, G.dinic(*l, *t)); // 添加树边
  fill(G.v, G.v + G.n + 1, false); G.dfscut(*l); // 求割集
  auto m = partition(l, r, [](int x){return G.v[x];});
  build(l, m); build(m, r);
}
```

2.8 Hopcroft mathcing

```
// 左侧 N 个点，右侧 K 个点 ， 1-based, 初始化将
  ↪ matx[],maty[] 都置为 0
int N, K, que[N], dx[N], dy[N], matx[N], maty[N];
int BFS() {
  int flag = 0, qt = 0, qh = 0;
  for(int i = 1; i <= K; ++ i) dy[i] = 0;
  for(int i = 1; i <= N; ++ i) {
    dx[i] = 0;
    if (! matx[i]) que[qt ++] = i;
  }
  while (qh < qt) {
    int u = que[qh ++];
    for(Edge *e = E[u]; e; e = e->n)
```

```cpp
      if (! dy[e->t]) {
        dy[e->t] = dx[u] + 1;
        if (! maty[e->t]) flag = true;
        else {
          dx[maty[e->t]] = dx[u] + 2;
          que[qt ++] = maty[e->t];
        }
      }
  }
  return flag;
}
int DFS(int u) {
  for(Edge *e = E[u]; e; e = e->n)
    if (dy[e->t] == dx[u] + 1) {
      dy[e->t] = 0;
      if (! maty[e->t] || DFS(maty[e->t])) {
          matx[u] = e->t; maty[e->t] = u; return true;
      }
    }
  return false;
}
void Hopcroft() {
  while (BFS()) for(int i = 1; i <= N; ++ i) if (! matx[i]) DFS(i);
}
```

2.9 KM

```cpp
// O(n^3), 0 base, 最大权匹配
// 不存在的边权值开到 -n * (|MAXV| + 1), INF 为 3n * (|MAXV| + 1)
int n, cost[N][N]; bool vy[N];
int lx[N], ly[N], match[N], slack[N], pre[N];
void augment(int root) {
  fill(vy + 1, vy + n + 1, false);
  fill(slack + 1, slack + n + 1, INF);
  int py; match[py = 0] = root;
  do { vy[py] = true; int x = match[py], delta = INF, yy;
    for (int y = 1; y <= n; y++) if (!vy[y]) {
      if (lx[x] + ly[y] - cost[x][y] < slack[y]) {
        slack[y] = lx[x] + ly[y] - cost[x][y];
        pre[y] = py;
      }
      if (slack[y] < delta) {
        delta = slack[y];
        yy = y;
      }
    }
    for (int y = 0; y <= n; y++) {
      if (vy[y]) {
        lx[match[y]] -= delta;
        ly[y] += delta;
      } else slack[y] -= delta;
    } py = yy;
  } while (match[py] != -1);
  do { int prev = pre[py];
    match[py] = match[prev];
    py = prev;
  } while (py);
}
void KM() {
  for (int i = 1; i <= n; i++) {
    lx[i] = ly[i] = 0; match[i] = -1;
    for (int j = 1; j <= n; j++)
      lx[i] = max(lx[i], cost[i][j]);
  }
  for (int root = 1; root <= n; root++) augment(root);
  // answer = \sum_i lx[i] + ly[i]
}
```

2.10 Maximum Clique

```cpp
const int N = 1000 + 7;
vector<vector<bool> > adj;
class MaxClique {
    const vector<vector<bool> > adj;
    const int n;
    vector<int> result, cur_res;
    vector<vector<int> > color_set;
    const double t_limit; // MAGIC
  int para, level;
  vector<pair<int, int> > steps;
public:
    class Vertex {
    public:
        int i, d;
        Vertex(int i, int d = 0) : i(i), d(d) {}
    };
    void reorder(vector<Vertex> &p) {
        for (auto &u : p) {
            u.d = 0;
            for (auto v : p) u.d += adj[v.i][u.i];
        }
        sort(p.begin(), p.end(), [&](const Vertex &a, const Vertex &b) { return a.d > b.d; } );
    }
  // reuse p[i].d to denote the maximum possible clique for first i vertices.
    void init_color(vector<Vertex> &p) {
        int maxd = p[0].d;
        for (int i = 0; i < p.size(); i++) p[i].d = min(i, maxd) + 1;
    }
    bool bridge(const vector<int> &s, int x) {
        for (auto v : s) if (adj[v][x]) return true;
        return false;
    }
  // approximate estimate the p[i].d
  // Do not care about first mink color class (For better result, we must get some vertex in some color class larger than mink )
    void color_sort(vector<Vertex> &cur) {
        int totc = 0, ptr = 0, mink = max((int)result.size() - (int)cur_res.size(), 0);
        for (int i = 0; i < cur.size(); i++) {
            int x = cur[i].i, k = 0;
            while (k < totc && bridge(color_set[k], x)) k++;
            if (k == totc) color_set[totc++].clear();
            color_set[k].push_back(x);
            if (k < mink) cur[ptr++].i = x;
        }
        if (ptr) cur[ptr - 1].d = 0;
        for (int i = mink; i < totc; i ++) {
            for (auto v : color_set[i]) {
                cur[ptr++] = Vertex(v, i + 1);
            }
        }
    }
    void expand(vector<Vertex> &cur) {
    steps[level].second = steps[level].second - steps[level].first + steps[level - 1].first;
    steps[level].first = steps[level - 1].second;
        while (cur.size()) {
            if (cur_res.size() + cur.back().d <= result.size()) return ;
            int x = cur.back().i;
            cur_res.push_back(x); cur.pop_back();
            vector<Vertex> remain;
            for (auto v : cur) {
                if (adj[v.i][x]) remain.push_back(v.i);
            }
```

```
            if (remain.size() == 0) {
                if (cur_res.size() > result.size()) result = cur_res;
            } else {
        // Magic ballance.
        if (1. * steps[level].second / ++para < t_limit) reorder(remain);
                color_sort(remain);
        steps[level++].second++;
                expand(remain);
        level--;
            }
            cur_res.pop_back();
        }
    }
public:
    MaxClique(const vector<vector<bool> > &_adj, int n, double tt = 0.025) : adj(_adj), n(n), t_limit(tt) {
        result.clear();
        cur_res.clear();
        color_set.resize(n);
    steps.resize(n + 1);
    fill(steps.begin(), steps.end(), make_pair(0, 0));
    level = 1;
    para = 0;
    }
    vector<int> solve() {
        vector<Vertex> p;
        for (int i = 0; i < n; i++) p.push_back(Vertex(i));
        reorder(p);
        init_color(p);
        expand(p);
        return result;
    }
};
```

2.11 原始对偶费用流

```
const LL INF = 1e18;
struct Edge { LL f, c; int to, r; };
vector<Edge> G[N];
int S, T, prv[N], prp[N], cur[N], vst[N];
LL d[N];
bool fst = true;
bool SPFA(int S) {
  if(fst){
    fst = 0;
    // 此处为第一次求最短路，可 Dij 就和下面一样，不可就 SPFA 或根据图性质 DP
    // ...
    return d[T] != INF;
  }else { // 此处为 Dij
    fill(d + 1, d + 1 + T, INF);
    priority_queue<pair<LL, int> > pq;
    pq.push({0, S});
    d[S] = 0;
    while(1) {
      while(!pq.empty() && -pq.top().first != d[pq.top().second]) pq.pop();
      if(pq.empty()) break;
      int v(pq.top().second); pq.pop();
      int cnt(0);
      for (Edge e : G[v]) {
        if (e.f && d[e.to] > d[v] + e.c) {
          d[e.to] = d[v] + e.c; prv[e.to] = v;
          prp[e.to] = cnt; pq.push({-d[e.to], e.to});
        }
        cnt++;
      }
    }
    return d[T] != INF;
  }
}
LL aug(int v, LL flow) { // 这里是多路增广才要抄的
    if(v == T) return flow;
    vst[v] = 1; LL flow1(flow);
    for(int & i(cur[v]); i < (int)G[v].size(); i++) {
        Edge & e = G[v][i];
        if(e.f && d[v] + e.c == d[e.to] && !vst[e.to]) {
            LL flow1(aug(e.to, min(flow, e.f)));
            flow -= flow1; e.f -= flow1;
            G[e.to][e.r].f += flow1;
        }
        if(flow == 0) {
            vst[v] = 0; return flow1 - flow;
        }
    }
    return flow1 - flow;
}
LL mcmf() {
    LL ans = 0, sT = 0;
    while (SPFA(S)) {
        sT += d[T]; // 这里是多路增广
        for(int i(1); i <= T; i++) cur[i] = 0, vst[i] = 0;
        ans += sT * aug(S, INF);
        /*LL f = INF; // 这里是单路增广
         for (int v = T; v != S; v = prv[v]) {
         int u = prv[v];        int j = prp[v];
         f = min(f, G[u][j].f);
         } for (int v = T; v != S; v = prv[v]) {
         int u = prv[v];        int j = prp[v];
         G[u][j].f -= f;        G[v][G[u][j].r].f += f;
         } sT += d[T]; ans += f * sT;*/
        for(int i(1); i <= T; i++)
            for(auto & e : G[i])
                e.c += d[i] - d[e.to];
    } return ans;
}
void add(int u, int v, int f, int c) {
  G[u].push_back({f, c, v, (int) G[v].size()});
  G[v].push_back({0, -c, u, (int) G[u].size() - 1});
}
int main() {
  // 初始化 S, T, T 编号最大, 1base
    // add(x, y, cap, cost)
  LL ans = mcmf();
}
```

2.12 完美消除序列

```
vector<int> adj[N], lst[N]; int rk[N], deg[N], tim[N], stmp, n;
vector<int> mcs(int n) {
  fill(deg + 1, deg + n + 1, 0);
  fill(rk + 1, rk + n + 1, 0);
  for (int i = 1; i <= n; i ++) lst[0].push_back(i);
  int ptr(0);
  for (int i = n; i >= 1; i --) {
    int p;
    for(; ;) {
      while (lst[ptr].empty()) ptr--;
      if (rk[lst[ptr].back()]) lst[ptr].pop_back();
      else {
        p = lst[ptr].back(); lst[ptr].pop_back(); break;
      }
    }
    rk[p] = i;
    for(int i : adj[p]) if(!rk[i]) {
      ptr = max(ptr, ++deg[i]);
      lst[deg[i]].push_back(i);
    }
  }
  vector<int> ret(n);
```

```cpp
  for(int i = 1; i <= n; i ++) ret[rk[i] - 1] = i;
  return ret;
} // 点从1开始标号, n 为点数, adj 为边表
int main() {
  static vector<vector<int> > chk[N];
  for(int i(0); i <= n; i++) adj[i].clear(), chk[i].clear(), lst[i].clear();
  vector<int> ord = mcs(n); // ord 是完美消除序列当且仅当原图是弦图
  for(int i(0); i < n; i++) {
    int v(ord[i]);
    vector<int> c;
    int mn(n);
    for(int y : adj[v]) if(rk[y] > rk[v]) {
      c.push_back(y);
      mn = min(mn, rk[y]);
    }
    chk[mn - 1].push_back(vector<int>());
    for(int y : c) if(rk[y] > mn) chk[mn - 1].back().push_back(y);
  }
  bool ok(true);
  for(int i(0); i < n && ok; i++) {
    int v(ord[i]);
    ++stmp;
    for(int y : adj[v]) tim[y] = stmp;
    for(int j(0); j < (int)chk[i].size() && ok; j++)
      for(int k(0); k < (int)chk[i][j].size() && ok; k++)
        if(tim[chk[i][j][k]] != stmp)
          ok = false;
  }
  assert(ok); // ok 代表是弦图 最小染色数只要从后往前贪心
}
```

2.13 Tarjan

```cpp
int dfn[N], low[N], tot_color, ins[N], color[N];
vector<int> adj[N], stk;
// 无向图割点, 割边, 边双连通分量
int tarjan(int u, int from) {
  static int tot = 0;
  low[u] = dfn[u] = ++tot;
  stk.push_back(u);
  for (auto v : adj[u]) {
    if (v == from) continue; // 有重边的话, 要判断不能走来的时候的边
    low[u] = min(low[u], dfn[v] ? dfn[v] : tarjan(v, u));
    // low[v] > dfn[u] ==> u <-> v 为割边
    // low[v] >= dfn[u] 且 u 不为根, 则 u 为割点
  }
  // 若 u 为根, 且至少有两个孩子 v1, v2, 满足 low[v1,v2] >= dfn[u], 则根为割点
  // 如果不用求边双连通分量, 可以去掉 stk 部分和之后的弹栈部分
  if (low[u] == dfn[u]) {
    int t; ++tot_color;
    do { t = stk.back(); stk.pop_back();
      color[t] = tot_color; ins[t] = false;
    } while (t != u);
  } return low[u];
}
// 无向图点双连通分量, 注意有向图求不了
// dfn 一开始需要赋值为 0
vector<vector<pair<int, int>> > bcc;
vector<pair<int, int>> stk;
int tarjan(int u, int fu) {
  static int tot = 0;
  low[u] = dfn[u] = ++tot;
  for (auto v : adj[u]) {
    if (v == fu) continue;
    if (dfn[v] < dfn[u]) stk.push_back({u, v});
    if (!dfn[v]) {
      low[u] = min(low[u], tarjan(v, u));
      if (low[v] >= dfn[u]) {
        bcc.push_back({});
        do { bcc.back().push_back(stk.back());
          stk.pop_back();
        } while (bcc.back().back() != make_pair(u, v));
      }
    } else low[u] = min(low[u], dfn[v]);
  } return low[u];
}
```

2.14 ZKW 费用流

```cpp
const int N = 105 << 2, M = 205 * 205 * 2;
const int inf = 1000000000;
int n, m, S, T, totFlow, totCost;
int dis[N], slack[N], visit[N];
/* vertices indexed from 1 to T */
int modlable() {
  int delta = inf;
  for(int i = 1; i <= T; i++) {
    if (!visit[i] && slack[i] < delta) delta = slack[i];
    slack[i] = inf;
  }
  if (delta == inf) return 1;
  for(int i = 1; i <= T; i++) if (visit[i]) dis[i] += delta;
  return 0;
}

int dfs(int x, int flow) {
  if (x == T) {
    totFlow += flow;
    totCost += flow * (dis[S] - dis[T]);
    return flow;
  }
  visit[x] = 1;
  int left = flow;
  for(int i = e.last[x]; ~i; i = e.succ[i]) if (e.cap[i] > 0 && !visit[e.other[i]]) {
    int y = e.other[i];
    if (dis[y] + e.cost[i] == dis[x]) {
      int delta = dfs(y, min(left, e.cap[i]));
      e.cap[i] -= delta;
      e.cap[i ^ 1] += delta;
      left -= delta;
      if (!left) { visit[x] = false; return flow; }
    } else {
      slack[y] = min(slack[y], dis[y] + e.cost[i] - dis[x]);
    }
  }
  return flow - left;
}
pair<int, int> minCost() {
  totFlow = 0, totCost = 0;
  fill(dis + 1, dis + T + 1, 0);
  do {
    do {
      fill(visit + 1, visit + T + 1, 0);
    } while(dfs(S, inf));
  } while(!modlable());
  return make_pair(totFlow, totCost);
}
int main() {
  e.clear();
}
```

3. String

3.1 Exkmp

```
// 如果想求一个字符串相对另外一个字符串的最长公共前缀，可以把他们拼接起来从而求得
void exkmp(char *s, int *a, int n) {
  a[0] = n; int p = 0, r = 0;
  for (int i = 1; i < n; ++i) {
    a[i] = (r > i) ? min(r - i, a[i - p]) : 0;
    while (i + a[i] < n && s[i + a[i]] == s[a[i]]) ++a[i];
    if (r < i + a[i]) r = i + a[i], p = i;
}}
```

3.2 Lyndon Word Decomposition

```
// 把串 s 划分成 lyndon words□ s1, s2, s3, ..., sk
// 每个串都严格小于他们的每个后缀，且串大小不增
// 如果求每个前缀的最小后缀，取最后一次 k 经过这个前缀的右边界时的信息更新
// 如果求每个前缀的最大后缀，更改大小于号，并且取第一次 k 经过这个前缀的信息更新
void lynDecomp() {
  vector<string> ss;
  for (int i = 0; i < n; ) {
    int j = i, k = i + 1; //mnsuf[i] = i;
    for (; k < n && s[k] >= s[j]; k++) {
      if (s[k] == s[j]) j++; // mnsuf[k] = mnsuf[j] + k - j;
      else j = i; // mnsuf[k] = i;
    }
    for (; i <= j; i += k - j) ss.push_back(s.substr(i, k - j));
  }
}
```

3.3 Manacher

```
// 这段代码仅仅处理奇回文，使用时请往字符串中间加入 # 来使用
for(int i = 1, j = 0; i != (n << 1) - 1; ++i){
  int p=i>>1, q = i - p, r = ((j + 1) >> 1) + l[j] - 1;
  l[i] = r < q ? 0 : min(r - q + 1, l[(j << 1) - i]);
  while (p - l[i] != -1 && q + l[i] != n
    && s[p - l[i]] == s[q + l[i]]) l[i]++;
  if(q + l[i] - 1 > r) j=i;
  a += l[i];
}
```

3.4 Minimum Representation

```
std::string find(std::string s) {
  int i, j, k, l, n = s.length(); s += s;
  for(i = 0, j = 1; j < n;) {
    for (k = 0; k < n && s[i + k] == s[j + k]; k++);
    if (k >= n) break;
    if (s[i + k] < s[j + k]) j += k + 1; // 如果求最大表示，换成 '>'
    else l = i + k, i = j, j = max(l, j) + 1;
  }
  return s.substr(i, n); // 可以通过求循环节来得到所有位置
}
```

3.5 Palindromic Automaton

```
struct node {
    node *child[C], *fail;
    int length; //cnt
    node(int length) : fail(NULL), length(length)
    {memset(child, NULL, sizeof(child));}
};
int size, text[N];
node *odd, *even;
node *match(node *now) {
    for (; text[size - now->length - 1] != text[size]; now = now->fail);
    return now;
}
bool extend(node *&last, int token) {
    text[++ size] = token;
    node *now = match(last);
    if (now->child[token])
        return last = now->child[token], false;
    last = now->child[token] = new node(now->length + 2);
    if (now == odd) last->fail = even;
    else {
        now = match(now->fail);
        last->fail = now->child[token];
    }
    //last -> cnt ++;
    return true;
}
void build() {
    text[size = 0] = -1;
    even = new node(0), odd = new node(-1);
    even->fail = odd;
}
// for in reversed ordered : x -> fail -> cnt += x -> cnt
```

3.6 Suffix Array

```
// unnecessary to double the array size or append 0 to the end.
// the string and the rank are 0 base.
int rk[N], height[N], sa[N];
int cmp(int *x,int a,int b,int d){
  return x[a]==x[b]&&x[a+d]==x[b+d];
}
void doubling(int *a,int n,int m){
  static int sRank[N],tmpA[N],tmpB[N];
  int *x=tmpA,*y=tmpB;
  for(int i=0;i<m;++i) sRank[i]=0;
  for(int i=0;i<n;++i) ++sRank[x[i]=a[i]];
  for(int i=1;i<m;++i) sRank[i]+=sRank[i-1];
  for(int i=n-1;i>=0;--i) sa[--sRank[x[i]]]=i;
  for(int d=1,p=0;p<n;m=p,d<<=1){
    p=0; for(int i=n-d;i<n;++i) y[p++]=i;
    for(int i=0;i<n;++i) if(sa[i]>=d) y[p++]=sa[i]-d;
    for(int i=0;i<m;++i) sRank[i]=0;
    for(int i=0;i<n;++i) ++sRank[x[i]];
    for(int i=1;i<m;++i) sRank[i]+=sRank[i-1];
    for(int i=n-1;i>=0;--i) sa[--sRank[x[y[i]]]]=y[i];
    swap(x,y); x[sa[0]]=0; p=1;
    y[n] = -1;
    for(int i=1;i<n;++i) x[sa[i]]=cmp(y,sa[i],sa[i-1],d)?p-1:p++;
  }
}
void calcHeight(int *a, int n){
  for(int i=0;i<n;++i) rk[sa[i]]=i;
  int cur=0; for(int i=0;i<n;++i)
  if(rk[i]){
    if(cur) cur--;
    for(;a[i+cur]==a[sa[rk[i]-1]+cur];++cur);
    height[rk[i]]=cur;
  }
}
```

3.7 Suffix Automaton

```
struct State {
    int len;
    State * parent, * go[2];
```

```
    State(int len = 0) : len(len), parent(NULL) {
        memset(go, 0, sizeof(go));
    }
    State * extend(State * , int token);
} node_pool[N * 2], *tot_node, *null = new State();
State * State::extend(State * start, int token) {
    State * p = this;
    State * np = this->go[token] ? null : new (tot_node++)
        ↪ State(this->len + 1);
    while(p && !p->go[token])
        p->go[token] = np, p = p->parent;
    if(!p) np->parent = start;
    else {
        State * q = p->go[token];
        if(p->len + 1 == q->len) {
            np->parent = q;
        } else {
            State * nq = new (tot_node++) State(*q);
            nq->len = p->len + 1;
            np->parent = q->parent = nq;
            while(p && p->go[token] == q) {
                p->go[token] = nq, p = p->parent;
            }
        }
    }
    return np == null ? np->parent : np;
}
void prepare() {
  tot_node = node_pool;
  head = tail = new(tot_node++) State();
  tail = tail->extend(head, token);  // to add one token
}
```

3.8 AC 自动机

```
struct node { node *ch[C], *fail; int cnt;
    node() { memset(ch, NULL, sizeof(ch));
            fail = NULL; cnt = 0; }
} pol[N], *tot = pol, *root;
node* newnode() { *tot = node(); return tot ++; }
void insert(char *t) {
    int n = strlen(t); node *p = root;
    for (int i = 0; i < n; ++ i) {
        int v = val(t[i]);
        if (!p->ch[v]) p->ch[v] = newnode();
        p = p->ch[v]; } p->cnt ++;
}
void BFS() {
    root->fail = root; queue<node*> q;
    q.push(root->fail = root);
    while (!q.empty()) {
        node* x = q.front(); q.pop();
        for (int i = 0; i < C; ++ i) if (x->ch[i]) {
            node *&y = x -> ch[i];
            y->fail = x == root ? root : x->fail->ch[i];
            y->cnt += x->ch[i]->fail->cnt; // 视情况
            q.push(y);
        } else x->ch[i]= x==root?root : x->fail->ch[i];
    }
} // root = newnode();
```

3.9 子串最长公共子序列

```
const int N = 2005;
int H[N][N], V[N][N];
char s[N], t[N];
int main() {
  gets(s + 1); gets(t + 1);
  int n = (int) strlen(s + 1);
  int m = (int) strlen(t + 1);
  for (int i = 1; i <= m; ++ i) H[0][i] = i;
  for (int i = 1; i <= n; ++ i) {
    for (int j = 1; j <= m; ++ j) {
      if (s[i] == t[j]) {
        H[i][j] = V[i][j - 1];
        V[i][j] = H[i - 1][j];
      } else {
        H[i][j] = max(H[i - 1][j], V[i][j - 1]);
        V[i][j] = min(H[i - 1][j], V[i][j - 1]);
  }}}
  for (int i = 1; i <= m; ++ i) {
    int ans = 0;
    for (int j = i; j <= m; ++ j) {
      ans += H[n][j] < i;
      printf("%d%c", ans, " \n"[j == m]);
    }
  }
}
```

4. Tree

4.1 树点分治-斜率优化

```
bool ena[mxn]; int s[mxn]; // s[x] 是子树 x 的大小
#define fore(i) for (auto i : G[x]) if (!ena[i])
#define fors(i) fore(i) if (i != p)
int size(int x, int p)
{ s[x] = 1; fors(i) s[x] += size(i, x); return s[x]; }
pii core(int x, int p, int sx, vi &st) {
    st.push_back(x); fors(i) if (sx - s[i] < s[i])
        return core(i, x, sx, st); return {x, p}; }
void divide(int y) {
    vi path; int x, yi; tie(x,yi)=core(y,0,s[y],path);
    path.pop_back(); for (int i : path) s[i] -= s[x];
    ena[x] = true; if (x != y) divide(y); // work(yi)
    for (int j : path); // ... // 从 y 到 x 收集 dp 值
    ... // 更新 x 的 dp 值并收集 更新注意复杂度
    fore(i) if (i != yi) work(i); // 更新 i 子树 ( 二分 )
    fore(i) if (i != yi) divide(i); ena[x] = false;
}
```

4.2 树链剖分

```
#define fors(i) for (auto i : e[x]) if (i != p)
int cnt; ai s, h, top, pa, dfn /*,hea*/;
int size(int x, int p)
{ s[x] = 1; fors(i) s[x] += size(i, x); return s[x]; }
void dfs(int x, int p, int t) {
  pa[x] = p, top[x] = t, h[x] = h[p] + 1, dfn[x] = ++cnt;
  int y = 0; // int &y = hea[x] = 0;
  fors(i) if (s[y] < s[i]) y = i;
  if (y) dfs(y, x, t);
  fors(i) if (i != y) dfs(i, x, i);
}
void build() { size(1, 0); cnt = 0; dfs(1, 0, 1); }
void path(int x, int y) {
  while (top[x] != top[y])
    if (h[top[x]] >= h[top[y]]) {
      foo(dfn[top[x]], dfn[x]); x = pa[top[x]];
    } else { // swap(x, y); 边权无向时可改用这句
      foo(dfn[top[y]], dfn[y]); y = pa[top[y]];
    }
  if (dfn[x] < dfn[y]) foo(dfn[x], dfn[y]);
  else foo(dfn[y], dfn[x]); // 边权时注意开闭
}
void subtree(int x) { foo(dfn[x], dfn[x] + s[x] - 1); }
```

4.3 虚树

```
// 点集并的直径端点 ⊂ 每个点集直径端点的并
// 可以用 dfs 序的 ST 表维护子树直径，建议使用 RMQLCA
void make(vi &poi) {
```

```cpp
    //poi 要按 dfn 排序 需要清空边表 E 注意 V 无序
    //0 号点相当于一个虚拟的根，需要 lca(u,0)==0,h[0]=0
    V = {0}; vi st = {0};
    for (int v : poi) {
        V.pb(v);int w=lca(st.back(),v), sz=st.size();
        while (sz > 1 && h[st[sz - 2]] >= h[w])
            E[st[sz - 2]].pb(st[sz - 1]), sz --;
        st.resize(sz);
        if (st[sz - 1] != w)
            E[w].pb(st.back()), st.back() = w, V.pb(w);
        st.pb(v);
    }
    for (int i=1; i<st.size(); ++i) E[st[i-1]].pb(st[i]);
}
```

4.4 有根树同构

```cpp
// O(1) 求逆 时间复杂度 O(n) MOD 需要是质数
#define fors(i) for (auto i : e[x]) if (i != p)
int ra[N]; void prepare() {
    for (int i = 0; i < N; ++ i) ra[i] = rand() % MOD;}
struct Sub {
    vector<int> s; int d1, d2, H1, H2;
    Sub() {d1 = d2 = 0; s.clear();}
    void add(int d, int v) { s.push_back(v);
        if (d>d1) d2=d1, d1=d; else if (d>d2) d2=d; }
    int hash() { H1 = H2 = 1; for (int i : s) {
        H1 = (ll) H1 * (ra[d1] + i) % MOD;
        H2 = (ll) H2 * (ra[d2] + i) % MOD;
    } return H1;
    }
    pii del(int d, int v) { if (d==d1)
        return {d2+1, (ll)H2*reverse(ra[d2]+v) % MOD};
        return {d1+1, (ll)H1*reverse(ra[d1]+v) % MOD};
    }
};
pii U[N]; int A[N]; Sub tree[N];
void dfsD(int x, int p) {
    tree[x] = Sub();
    fors(i) { dfsD(i, x);
        tree[x].add(tree[i].d1 + 1, tree[i].H1); }
    tree[x].hash();
}
void dfsU(int x, int p) {
    if (p) tree[x].add(U[x].first, U[x].second);
    A[x] = tree[x].hash();
    fors(i){U[i]=tree[x].del(tree[i].d1+1,tree[i].H1);
        dfsU(i, x); }
}
```

5. Math

5.1 Conclusions

```cpp
// ∏_{k=1,gcd(k,m)=1}^{m} k = -1 ( mod m ) if m = 4, p^q, 2p^q
// otherwise 1 ( mod m )
```

5.2 积性函数线性求法

```cpp
int main() {
  static int mu[N], is_prime[N];
  fill(is_prime, is_prime + MAXV, true);
  mu[1] = 1;
  vector<int> primes;
  for (int i = 2; i < MAXV; i++) {
    if (is_prime[i]) {
      primes.push_back(i); mu[i] = -1;
    }
    for (auto p : primes) {
      if (1LL * i * p >= MAXV) break;
      is_prime[p * i] = false;
      if (i % p == 0) {
        mu[i * p] = 0; break;
      } else { mu[i * p] = -mu[i]; }
    }
  }
  return 0;
}
```

5.3 平方剩余

```cpp
// x^2 = a (mod p), 0 <= a < p, 返回 true or false 代表是否存在解
// p 必须是质数，若是多个单次质数的乘积，可以分别求解再用 CRT 合并
// 复杂度为 O(log n)
void multiply(ll &c, ll &d, ll a, ll b, ll w) {
    int cc = (a * c + b * d % MOD * w) % MOD;
    int dd = (a * d + b * c) % MOD;
    c = cc, d = dd;
}

bool solve(int n, int &x) {
    if (MOD == 2) return x = 1, true;
    if (power(n, MOD / 2, MOD) == MOD - 1) return false;
    ll c = 1, d = 0, b = 1, a, w;
    // finding a such that a^2 - n is not a square
    do { a = rand() % MOD;
        w = (a * a - n + MOD) % MOD;
        if (w == 0) return x = a, true;
    } while (power(w, MOD / 2, MOD) != MOD - 1);
    for (int times = (MOD + 1) / 2; times; times >>= 1) {
        if (times & 1) multiply(c, d, a, b, w);
        multiply(a, b, a, b, w);
    }
    // x = (a + sqrt(w)) ^ ((p + 1) / 2)
    return x = c, true;
}
```

5.4 线性同余不等式

```cpp
// Find the minimal non-negtive solutions for l ≤ d · x mod m ≤ r
// 0 ≤ d, l, r < m; l ≤ r, O(log n)
ll cal(ll m, ll d, ll l, ll r) {
    if (l == 0) return 0;
    if (d == 0) return MXL; // 无解
    if (d * 2 > m) return cal(m, m - d, m - r, m - l);
    if ((l - 1) / d < r / d) return (l - 1) / d + 1;
    ll k = cal(d, (-m % d + d) % d, l % d, r % d);
    return k == MXL ? MXL : (k * m + l - 1) / d + 1; // 无解 2
}

// return all x satisfying l1<=x<=r1 and l2<=(x*mul+add)%LIM<=r2
// here LIM = 2^32 so we use UI instead of "%".
// O(log p + #solutions)
struct Jump {
    UI val, step;
    Jump(UI val, UI step) : val(val), step(step) { }
    Jump operator + (const Jump & b) const {
        return Jump(val + b.val, step + b.step); }
    Jump operator - (const Jump & b) const {
        return Jump(val - b.val, step + b.step);
    }};
inline Jump operator * (UI x, const Jump & a) {
    return Jump(x * a.val, x * a.step);
}
vector<UI> solve(UI l1, UI r1, UI l2, UI r2, pair<UI, UI> muladd) {
    UI mul = muladd.first, add = muladd.second, w = r2 - l2;
```

```
    Jump up(mul, 1), dn(-mul, 1);
    UI s(l1 * mul + add);
    Jump lo(r2 - s, 0), hi(s - l2, 0);
    function<void(Jump &, Jump &)> sub = [&](Jump & a, Jump & b) {
        if (a.val > w) {
            UI t(((long long)a.val - max(0ll, w + 1ll - b.val)) / b.val);
            a = a - t * b;
        }
    };
    sub(lo, up), sub(hi, dn);
    while (up.val > w || dn.val > w) {
        sub(up, dn); sub(lo, up);
        sub(dn, up); sub(hi, dn); }
    assert(up.val + dn.val > w);
    vector<UI> res;
    Jump bg(s + mul * min(lo.step, hi.step), min(lo.step, hi.step));
    while (bg.step <= r1 - l1) {
        if (l2 <= bg.val && bg.val <= r2)
            res.push_back(bg.step + l1);
        if (l2 <= bg.val - dn.val && bg.val - dn.val <= r2) {
            bg = bg - dn;
        } else bg = bg + up;
    } return res;
}
```

5.5 Schreier Sims

```
struct Perm{
  vector<int> P; Perm() {} Perm(int n) { P.resize(n); }
  Perm inv()const{
    Perm ret(P.size());
    for(int i = 0; i < int(P.size()); ++i) ret.P[P[i]] = i;
    return ret;
  }
  int &operator [](const int &dn){ return P[dn]; }
  void resize(const size_t &sz){ P.resize(sz); }
  size_t size()const{ return P.size(); }
  const int &operator [](const int &dn)const{ return P[dn]; }
};
Perm operator *(const Perm &a, const Perm &b){
  Perm ret(a.size());
  for(int i = 0; i < (int)a.size(); ++i) ret[i] = b[a[i]];
  return ret;
}
typedef vector<Perm> Bucket;
typedef vector<int> Table;
typedef pair<int,int> PII;
int n, m;
vector<Bucket> buckets, bucketsInv; vector<Table> lookupTable;
int fastFilter(const Perm &g, bool addToGroup = true) {
  int n = buckets.size();
  Perm p(g);
  for(int i = 0; i < n; ++i){
    int res = lookupTable[i][p[i]];
    if(res == -1){
      if(addToGroup){
        buckets[i].push_back(p); bucketsInv[i].push_back(p.inv());
        lookupTable[i][p[i]] = (int)buckets[i].size() - 1;
      }
      return i;
    }
    p = p * bucketsInv[i][res];
  }
  return -1;
}
long long calcTotalSize(){
  long long ret = 1;
  for(int i = 0; i < n; ++i) ret *= buckets[i].size();
  return ret;
}
bool inGroup(const Perm &g){ return fastFilter(g, false) == -1; }
void solve(const Bucket &gen,int _n){// m perm[0..n - 1]s
  n = _n, m = gen.size();
  {//clear all
    vector<Bucket> _buckets(n); swap(buckets, _buckets);
    vector<Bucket> _bucketsInv(n); swap(bucketsInv, _bucketsInv);
    vector<Table> _lookupTable(n); swap(lookupTable, _lookupTable);
  }
  for(int i = 0; i < n; ++i){
    lookupTable[i].resize(n);
    fill(lookupTable[i].begin(), lookupTable[i].end(), -1);
  }
  Perm id(n);
  for(int i = 0; i < n; ++i) id[i] = i;
  for(int i = 0; i < n; ++i){
    buckets[i].push_back(id); bucketsInv[i].push_back(id);
    lookupTable[i][i] = 0;
  }
  for(int i = 0; i < m; ++i) fastFilter(gen[i]);
  queue<pair<PII,PII> > toUpdate;
  for(int i = 0; i < n; ++i)
    for(int j = i; j < n; ++j)
      for(int k = 0; k < (int)buckets[i].size(); ++k)
        for(int l = 0; l < (int)buckets[j].size(); ++l)
          toUpdate.push(make_pair(PII(i,k), PII(j,l)));
  while(!toUpdate.empty()){
    PII a = toUpdate.front().first, b = toUpdate.front().second;
    toUpdate.pop();
    int res = fastFilter(buckets[a.first][a.second] * buckets[b.first][b.second]);
    if(res==-1) continue;
    PII newPair(res, (int)buckets[res].size() - 1);
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < (int)buckets[i].size(); ++j){
        if(i <= res) toUpdate.push(make_pair(PII(i, j), newPair));
        if(res <= i) toUpdate.push(make_pair(newPair, PII(i, j)));
      }
  }
}
```

5.6 CRT

```
inline void euclid(const LL & a, const LL &b, LL &x, LL &y) {
  if (b == 0) x = 1, y = 0;
  else euclid(b, a % b, y, x), y -= a / b * x;
}
void combine(LL r1, LL m1, LL &r2, LL &m2, LL d) {
  if(m1 > m2) swap(r1, r2), swap(m1, m2);
  LL x, y;
  euclid(m1, m2, x, y);
  m1 /= d;
  LL tmp((r1 - r2) / d * y % m1);
  if(tmp < 0) tmp += m1;
  r2 += tmp * m2;
  m2 *= m1;
}
inline bool crt(int n, const vector<LL> & r, const vector<LL> & m,
  LL & rem, LL & mod) {
```

```cpp
  rem = 0; mod = 1;
  for (int i = 0; i < (int)r.size(); ++i) {
    LL div(gcd(mod, m[i]));
    if ((r[i] - rem) % div) {
      return false;
    }
    combine(r[i], m[i], rem, mod, div);
  }
  return true;
}
```

5.7 Factorial Mod

```cpp
// Complexity is O(pq + q^2 log_2 p)
int calcsgn(LL x) { return (x % 8 <= 2 || x % 8 == 7) ? 1 : -1; } // 计算mod 4的答案
// 1 <= n <= 1000, p^q <= 1000 测试通过, fastpo 是 LL LL LL 参数
LL f(LL n, LL p, LL q) {
  LL mod(fastpo(p, q, INT64_MAX));
  LL phi(mod / p * (p - 1));
  static LL pre[1111111];
  pre[0] = 1;
  for(int i(1); i <= p * (q + 1); i++) pre[i] = i % p == 0 ? pre[i - 1] : pre[i - 1] * i % mod;
  LL res(1);
  LL u(n / p), v(n % p);
  for(int j(1); j < q; j++) {
    __int128 alpha(1);
    for(int i(j + 1); i < q; i++) alpha = alpha * (u - i) / (j - i);
    for(int i(j - 1); i >= 0; i--) alpha = alpha * (u - i) / (j - i);
    alpha = (alpha % phi + phi) % phi;
    res = res * fastpo(pre[j * p + v] % mod * fastpo(pre[v], phi - 1, mod) % mod * fastpo(pre[j * p], phi - 1, mod) % mod, alpha, mod) % mod;
  }
  int sgn(calcsgn(u * 2));
  int r(max((LL)1, q / 2 + 1));
  for(int j(1); j <= r; j++) {
    __int128 beta(1);
    for(int i(j + 1); i <= r; i++) beta = beta * (u - i) / (j - i);
    for(int i(j - 1); i > -j; i--) beta = beta * (u - i) / (j - i);
    beta *= u + j;
    for(int i(-j - 1); i >= -r; i--) beta = beta * (u - i) / (j - i);
    assert(beta % (j + u) == 0);
    beta /= u + j;
    beta = (beta % phi + phi) % phi;
    if(beta % 2)
      sgn *= calcsgn(j * 2);
    res = res * fastpo(pre[j * p], beta, mod) % mod;
  }
  if(p == 2) res = (res * sgn + mod) % mod;
  res = res * pre[v] % mod;
  return res;
}
```

5.8 Miller Rabin and Pollard Rho

```cpp
bool miller_rabin(long long n, int base) {
  long long n2 = n - 1, s = 0;
  while (~n2 & 1) n2 >>= 1, s++;
  long long ret = powmod(base, n2, n);
  if (ret == 1 || ret == n - 1) return true;
  for (s--; s >= 0; s--) {
  if ((ret = mulmod(ret, ret, n)) == n - 1) return true;
  }
    return false; // n is not a strong pseudo prime
}
bool isprime(long long n) {
  static long long base[] = {2,3,5,7,11,13,17,19,23,29,31,37};
  static long long lim[] = {4, 0, 1373653LL, 25326001LL,
        25000000000LL, 2152302898747LL, 3474749660383LL,
        341550071728321LL, 0, 0, 0, 0};
  if (n < 2 || n == 3215031751LL) return 0;
  for(int i = 0; i < 12 && base[i] < n; ++i) {
    if (n < lim[i]) return true;
    if (!miller_rabin(n, base[i])) return false;
  }
  return true;
}
long long f(long long x, long long m) { return (mulmod(x, x, m) + 1) % m; }
long long rho(long long n) {
  if (n == 1 || isprime(n)) return n;
  if (n % 2 == 0) return 2;
  for (int i = 1; ; i ++) {
    long long x = i, y = f(x, n), p = __gcd(y - x, n);
    while (p == 1) { x = f(x, n); y = f(f(y, n), n);
      p = __gcd((y - x + n) % n, n) % n;
    }
    if (p != 0 && p != n) return p;
}} // 分解时需特判 n = 1
```

5.9 Pell 方程

```cpp
// x_{k+1} = x_0 x_k + n y_0 y_k
// y_{k+1} = x_0 y_k + y_0 x_k
pair<ll, ll> pell(ll n) {
  static ll p[N], q[N], g[N], h[N], a[N];
  p[1] = q[0] = h[1] = 1; p[0] = q[1] = g[1] = 0;
  a[2] = (ll)(floor(sqrtl(n) + 1e-7L));
  for(int i = 2; ; i ++) {
    g[i] = -g[i - 1] + a[i] * h[i - 1];
    h[i] = (n - g[i] * g[i]) / h[i - 1];
    a[i + 1] = (g[i] + a[2]) / h[i];
    p[i] = a[i] * p[i - 1] + p[i - 2];
    q[i] = a[i] * q[i - 1] + q[i - 2];
    if(p[i] * p[i] - n * q[i] * q[i] == 1)
      return {p[i], q[i]};
}} // x^2 - n * y^2 = 1 最小正整数根, n 为完全平方数时无解
```

5.10 Simplex

```cpp
// 求max{cx | Ax <= b, x >= 0}的解
typedef vector<double> VD;
VD simplex(vector<VD> A, VD b, VD c) {
  int n = A.size(), m = A[0].size() + 1, r = n, s = m - 1;
  vector<VD> D(n + 2, VD(m + 1, 0)); vector<int> ix(n + m);
  for (int i = 0; i < n + m; ++ i) ix[i] = i;
  for (int i = 0; i < n; ++ i) {
    for (int j = 0; j < m - 1; ++ j) D[i][j] = -A[i][j];
    D[i][m - 1] = 1; D[i][m] = b[i];
    if (D[r][m] > D[i][m]) r = i;
  }
  for (int j = 0; j < m - 1; ++ j) D[n][j] = c[j];
  D[n + 1][m - 1] = -1;
  for (double d; ; ) {
    if (r < n) {
      int t = ix[s]; ix[s] = ix[r + m]; ix[r + m] = t;
      D[r][s] = 1.0 / D[r][s]; vector<int> speedUp;
      for (int j = 0; j <= m; ++ j) if (j != s) {
        D[r][j] *= -D[r][s];
        if(D[r][j]) speedUp.push_back(j);
      }
      for (int i = 0; i <= n + 1; ++ i) if (i != r) {
        for(int j = 0; j < speedUp.size(); ++ j)
```

```
      D[i][speedUp[j]] += D[r][speedUp[j]] * D[i][s];
      D[i][s] *= D[r][s];
    }} r = -1; s = -1;
    for (int j = 0; j < m; ++ j) if (s < 0 || ix[s] > ix[j])
      if (D[n + 1][j] > EPS || (D[n + 1][j] > -EPS && D[n][j] > EPS)) s = j;
    if (s < 0) break;
    for (int i = 0; i < n; ++ i) if (D[i][s] < -EPS)
      if (r < 0 || (d = D[r][m] / D[r][s] - D[i][m] / D[i][s]) < -EPS
        || (d < EPS && ix[r + m] > ix[i + m])) r = i;
    if (r < 0) return VD(); // 无边界
  }
  if (D[n + 1][m] < -EPS) return VD(); // 无解
  VD x(m - 1);
  for (int i = m; i < n + m; ++ i) if (ix[i] < m - 1) x[ix[i]] = D[i - m][m];
  return x; // 最优值在 D[n][m]
}

namespace simplex {
  const int N=410,M=30010;
  int n,m;
  int Left[M],Down[N],idx[N];
  ll a[M][N],b[M],c[N],v;
  void init(int p,int q) {
    n=p; m=q;
    rep(i,1,m+1) rep(j,1,n+1) a[i][j]=0;
    rep(j,1,m+1) b[j]=0; rep(i,1,n+1) c[i]=0;
    rep(i,1,n+1) idx[i]=0;
    v=0;
  }
  int va[N];
  void pivot(int x,int y) {
    swap(Left[x],Down[y]);
    ll k=a[x][y];
    a[x][y]=1; b[x]/=k;
    int t=0;
    rep(j,1,n+1) {
      a[x][j]/=k;
      if (a[x][j]) va[++t]=j;
    }
    rep(i,1,m+1) if(i!=x&&a[i][y]) {
      k=a[i][y];
      a[i][y]=0;
      b[i]-=k*b[x];
      rep(j,1,t+1) a[i][va[j]]-=k*a[x][va[j]];
    }
    k=c[y];
    c[y]=0;
    v+=k*b[x];
    rep(j,1,t+1) c[va[j]]-=k*a[x][va[j]];
  }
  int solve() {
    rep(i,1,n+1) Down[i]=i;
    rep(i,1,m+1) Left[i]=n+i;
    while(1) {
      int x=0;
      rep(i,1,m+1) if (b[i]<0) { x=i; break; }
      if(x==0) break;
      int y=0;
      rep(j,1,n+1) if (a[x][j]<0) { y=j; if(rand()&1) break; }
      if(y==0) { puts("Infeasible"); return -1; } //Infeasible
      pivot(x,y);
    }
    while(1) {
      int y=0;
      rep(i,1,n+1) if (c[i]>0&&(y==0||c[i]>c[y])) y=i;
      if(y==0) break;
      int x=0;
      rep(j,1,m+1) if (a[j][y]>0) if (x==0||b[j]/a[j][y]<b[x]/a[x][y]) x=j;
      if(x==0) { puts("Unbounded"); return -2; } // Unbounded
      pivot(x,y);
    }
    printf("%lld\n",v);
    rep(i,1,n+1) if(Left[i]<=n) idx[Left[i]]=i;
    rep(i,1,n+1) printf("%lld ",b[idx[i]]);
    puts("");
    return 1;
  }
}
```

5.11 Simpson

```
// 三次函数，两倍精度拟合
// error = $\frac{(r-l)^5}{6480}|f^{(4)}|$
// $\int_a^b f(x)\,dx \approx \frac{(b-a)}{8}\left[f(a)+3f\left(\frac{2a+b}{3}\right)+3f\left(\frac{a+2b}{3}\right)+f(b)\right]$
// 三次函数拟合 error = $\frac{1}{90}\left(\frac{(r-l)}{2}\right)^5|f^{(4)}|$
d simpson(d fl,d fr,d fmid,d l,d r) {
  return (fl+fr+4.0*fmid)*(r-l)/6.0; }
d rsimpson(d slr,d fl,d fr,d fmid,d l,d r) {
  d mid = (l+r)/2,fml = f((l+mid)/2),fmr = f((mid+r)/2);
  d slm = simpson(fl,fmid,fml,l,mid);
  d smr = simpson(fmid,fr,fmr,mid,r);
  if(fabs(slr - smr - slm) / slr < eps)return slm + smr;
  return rsimpson(slm,fl,fmid,fml,l,mid)+
    rsimpson(smr,fmid,fr,fmr,mid,r);
}
```

5.12 FFT

```
// double 精度对$10^9+7$ 取模最多可以做到$2^{20}$
const int MOD = 1000003;
const double PI = acos(-1);
typedef complex<double> Complex;
const int N = 65536, L = 15, MASK = (1 << L) - 1;
Complex w[N];
void FFTInit() {
  for (int i = 0; i < N; ++i)
    w[i] = Complex(cos(2 * i * PI / N), sin(2 * i * PI / N));
}
void FFT(Complex p[], int n) {
  for (int i = 1, j = 0; i < n - 1; ++i) {
    for (int s = n; j ^= s >>= 1, ~j & s;);
    if (i < j) swap(p[i], p[j]);
  }
  for (int d = 0; (1 << d) < n; ++d) {
    int m = 1 << d, m2 = m * 2, rm = n >> (d + 1);
    for (int i = 0; i < n; i += m2) {
      for (int j = 0; j < m; ++j) {
        Complex &p1 = p[i + j + m], &p2 = p[i + j];
        Complex t = w[rm * j] * p1;
        p1 = p2 - t, p2 = p2 + t;
      } } }
}
Complex A[N], B[N], C[N], D[N];
void mul(int a[N], int b[N]) {
  for (int i = 0; i < N; ++i) {
    A[i] = Complex(a[i] >> L, a[i] & MASK);
    B[i] = Complex(b[i] >> L, b[i] & MASK);
  }
  FFT(A, N), FFT(B, N);
  for (int i = 0; i < N; ++i) {
    int j = (N - i) % N;
    Complex da = (A[i] - conj(A[j])) * Complex(0, -0.5),
        db = (A[i] + conj(A[j])) * Complex(0.5, 0),
```

```
            dc = (B[i] - conj(B[j])) * Complex(0, -0.5),
            dd = (B[i] + conj(B[j])) * Complex(0.5, 0);
        C[j] = da * dd + da * dc * Complex(0, 1);
        D[j] = db * dd + db * dc * Complex(0, 1);
    }
    FFT(C, N), FFT(D, N);
    for (int i = 0; i < N; ++i) {
        long long da = (long long)(C[i].imag() / N + 0.5) % MOD,
            db = (long long)(C[i].real() / N + 0.5) % MOD,
            dc = (long long)(D[i].imag() / N + 0.5) % MOD,
            dd = (long long)(D[i].real() / N + 0.5) % MOD;
        a[i] = ((dd << (L * 2)) + ((db + dc) << L) + da) % MOD;
    }
}
```

$$A(x)B(x) \equiv 1 \pmod{x^n} \tag{1}$$

$$(A(x)B(x) - 1)^2 \equiv 0 \pmod{x^{2n}} \tag{2}$$

$$A(x)(2B(x) - B(x)^2 A(x)) \equiv 1 \pmod{x^{2n}} \tag{3}$$

$$B(x) = \ln A(x) \tag{4}$$

$$B'(x) = \frac{A'(x)}{A(x)} \tag{5}$$

$$f(x) = \exp A(x) \tag{6}$$

$$g(f(x)) = \ln f(x) - A(x) = 0 \tag{7}$$

$$f(x) \equiv f_0(x) \pmod{x^n} \tag{8}$$

$$f(x) \equiv f_0(x)(1 - \ln f_0(x) + A(x)) \pmod{x^{2n}} \tag{9}$$

5.13 解一元三次方程

```
double a(p[3]), b(p[2]), c(p[1]), d(p[0]);
double k(b / a), m(c / a), n(d / a);
double p(-k * k / 3. + m);
double q(2. * k * k * k / 27 - k * m / 3. + n);
Complex omega[3] = {Complex(1, 0), Complex(-0.5, 0.5 * sqrt(3)), Complex(-0.5, -0.5 * sqrt(3))};
Complex r1, r2;
double delta(q * q / 4 + p * p * p / 27);
if (delta > 0) {
    r1 = cubrt(-q / 2. + sqrt(delta));
    r2 = cubrt(-q / 2. - sqrt(delta));
} else {
    r1 = pow(-q / 2. + pow(Complex(delta), 0.5), 1. / 3);
    r2 = pow(-q / 2. - pow(Complex(delta), 0.5), 1. / 3);
}
for(int _(0); _ < 3; _++) {
    Complex x = -k / 3. + r1 * omega[_ * 1] + r2 * omega[_ * 2 % 3];
}
```

5.14 线性递推

```
// Calculating kth term of linear recurrence sequence
// Complexity: init O(n^2log) query O(n^2logk)
// Requirement: const LOG const MOD
// Input(constructor): vector<int> - first n terms
//                     vector<int> - transition function
// Output(calc(k)): int - the kth term mod MOD
// Example: In: {1, 1} {2, 1} an = 2an-1 + an-2
//          Out: calc(3) = 3, calc(10007) = 71480733 (MOD 1e9+7)
struct LinearRec {
    int n;
    vector<int> first, trans;
    vector<vector<int> > bin;
vector<int> add(vector<int> &a, vector<int> &b) {
    vector<int> result(n * 2 + 1, 0);
    // 不要每次新开 vector，可以使用矩阵乘法优化
    for (int i = 0; i <= n; ++i) {
        for (int j = 0; j <= n; ++j) {
            result[i + j] += (long long)a[i] * b[j] % MOD;
            if (result[i + j] >= MOD) {
                result[i + j] -= MOD;
            }
        }
    }
    for (int i = 2 * n; i > n; --i) {
        for (int j = 0; j < n; ++j) {
            result[i - 1 - j] += (long long)result[i] * trans[j] % MOD;
            if (result[i - 1 - j] >= MOD)
                result[i - 1 - j] -= MOD;
        }
        result[i] = 0;
    }
    result.erase(result.begin() + n + 1, result.end());
    return result;
}
LinearRec(vector<int> &first, vector<int> &trans):first(first), trans(trans) {
    n = first.size();
    vector<int> a(n + 1, 0);
    a[1] = 1;
    bin.push_back(a);
    for (int i = 1; i < LOG; ++i)
        bin.push_back(add(bin[i - 1], bin[i - 1]));
}
int calc(int k) {
    vector<int> a(n + 1, 0);
    a[0] = 1;
    for (int i = 0; i < LOG; ++i)
        if (k >> i & 1)
            a = add(a, bin[i]);
    int ret = 0;
    for (int i = 0; i < n; ++i)
        if ((ret += (long long)a[i + 1] * first[i] % MOD) >= MOD)
            ret -= MOD;
    return ret;
}};
```

5.15 黑盒子代数

```
// Berlekamp-Massey Algorithm
// Complexity: O(n^2)
// Requirement: const MOD, inverse(int)
// Input: vector<int> - the first elements of the sequence
// Output: vector<int> - the recursive equation of the given sequence
// Example:  In: {1, 1, 2, 3}  Out: {1, 1000000006, 1000000006} (MOD = 1e9+7)
struct Poly {
    vector<int> a;
    Poly() { a.clear(); }
    Poly(vector<int> &a): a(a) {}
    int length() const { return a.size(); }
    Poly move(int d) {
        vector<int> na(d, 0);
        na.insert(na.end(), a.begin(), a.end());
        return Poly(na);
    }
    int calc(vector<int> &d, int pos) {
        int ret = 0;
        for (int i = 0; i < (int)a.size(); ++i) {
            if ((ret += (long long)d[pos - i] * a[i] % MOD) >= MOD) {
                ret -= MOD; }}
        return ret;
    }
```

```
    Poly operator - (const Poly &b) {
        vector<int> na(max(this->length(), b.length()));
        for (int i = 0; i < (int)na.size(); ++i) {
            int aa = i < this->length() ? this->a[i] : 0,
            bb = i < b.length() ? b.a[i] : 0;
            na[i] = (aa + MOD - bb) % MOD;
        }
        return Poly(na);
    }
};
Poly operator * (const int &c, const Poly &p) {
    vector<int> na(p.length());
    for (int i = 0; i < (int)na.size(); ++i) {
        na[i] = (long long)c * p.a[i] % MOD;
    }
    return na;
}
vector<int> solve(vector<int> a) {
    int n = a.size();
    Poly s, b;
    s.a.push_back(1), b.a.push_back(1);
    for (int i = 1, j = 0, ld = a[0]; i < n; ++i) {
        int d = s.calc(a, i);
        if (d) {
            if ((s.length() - 1) * 2 <= i) {
                Poly ob = b;
                b = s;
                s = s - (long long)d * inverse(ld) % MOD * ob.move(i - j);
                j = i;
                ld = d;
            } else {
                s = s - (long long)d * inverse(ld) % MOD * b.move(i - j);
            }
        }
    }
    //Caution: s.a might be shorter than expected
    return s.a;
}
/*
 如果要求行列式，只需要求出来特征多项式即可，
 而这个方法可以解出来最小多项式，如果最小多项式里面有 x 的因子，那么行列式必然为 0
 否则我们让原矩阵乘以一个随机的对角阵，那么高概率最小多项式次数为 n，那么也就是那个矩阵的
 特征多项式从而容易求得行列式 .
 */
```

6. Data Structure

6.1 可持久化左偏树-K短路

```
#define nil mem
struct Node { Node *l, *r, *s; int dist; Val val, laz;
} mem[mxv]={{nil,nil,nil,-1}}; int sz; using ptr = Node*;
#define NEW(arg...) new(mem + ++sz)Node{nil,nil,nil,0,arg}
#define COPY(x) new(mem + ++sz)Node(*(x))
ptr add(ptr x, Val ope) {
    if (x == nil) return nil;
    x = COPY(x); x->val += ope; x->laz += ope; return x; }
ptr down(ptr x) {
    if (x == nil) return nil; x = COPY(x); if (x->laz)
    { x->l = add(x->l, x->laz); x->r = add(x->r, x->laz);
        x->s = add(x->s, x->laz); x->laz = 0; } return x;}
ptr sub_merge(ptr x, ptr y) {
    if (x == nil) return y; if (y == nil) return x;
    if (cmp(y->val, x->val)) swap(x, y);
    x = down(x); x->r = sub_merge(x->r, y);
    if (x->l->dist < x->r->dist) swap(x->l, x->r);
    x->dist = x->r->dist + 1; return x; }
ptr merge(ptr x, ptr y) {
    if (x == nil) return y; if (y == nil) return x;
    if (cmp(y->val, x->val)) swap(x, y); // 小根堆 (less)
    x = down(x); x->s = sub_merge(x->s, y); return x; }
ptr pop(ptr x) { // pop 操作注意仔细计算复杂度
    x = down(x); x = x->s; x = down(x);
    x->s = sub_merge(x->s, sub_merge(x->l, x->r));
    x->l = x->r = nil; return x; }
/* Hint for K 短路 : 先建最短路树, d[x] 表示到 T 的距离
非树边的权值是比最短路多走的距离 . 一条路径用经过了
某些非树边表示□ 考虑每次可以从最后一条非树边的末端,
新长一条从末端到 T 的路径上出发的权值最小的非树边 ;
或者是删掉最后这条非树边 (pop), 用次小边替代 .
按照非树边的权值建堆 , 需要记录末端点 . 注意判断堆非空 .
priority_queue<dis,end point at where,heap ptr>
堆里的初值 : {d[S]+root[S].top.len, root[S].top.at, root[S]}
每次两种转移 : if ((root1 = pop(p.heap)) != nil)
{p.dis-p.heap.top.len+root1.top.len,(root1->val).at,root1}
if ((root2 = root[p.at]) != nil)
{p.dis + root2.top.len, (root2 -> val).at, root2} */
```

6.2 左偏树

```
#define nil mem
struct Node { Node *l, *r; int dist; Val val; }
    mem[mxv] = {{nil, nil, -1}}; int sz = 0;
#define NEW(arg...) (new(mem + ++sz)Node{nil,nil,0,arg})
//add(x,ope){if(x!=nil){x->val+=ope,x->laz+=ope;}}
//down(x) { if(x->laz) { add(x->l,x->laz);
//             add(x->r,x->laz);x->laz=0;} }
Node *merge(Node *x, Node *y) {
    if (x == nil) return y;
    if (y == nil) return x;
    if (y->val < x->val) swap(x, y); // 默认小根堆
    // down(x); // 朱刘算法下传标记预留位置
    x->r = merge(x->r, y);
    if (x->l->dist < x->r->dist) swap(x->l, x->r);
    x->dist = x->r->dist + 1;
    return x; }
Node *pop(Node *x) {/*down(x);*/return merge(x->l,x->r);}
```

6.3 KD 树

```
// 带插入版本 , 没有写内存回收 , 空间复杂度 n log n ,
// 如果不需要插入可以大大简化 N 为最大点数, D 为每个点的最大
// 维度 , d 为实际维度 以查找最近点为例 ret 为当前最近点的距离
// 的平方用来剪枝 , 查询 k 近或 k 远的方法类似 注意先 initnull
const ll INF = (int)1e9 + 10;
const int N = 2000000 + 10, D = 5;
const double SCALE = 0.75; int d;
struct poi { int x[D]; } buf[N];
long long dist(const poi &a, const poi &b) {...}
struct node {
    int dep, sz; node *ch[2], *p; poi val, maxv, minv;
    void set(node *t, int d) { ch[d] = t; t->p = this; }
    bool dir() { return this == p->ch[1]; }
    bool balanced(){return (double)max(ch[0]->sz,ch[1]->sz)
                                  <= (double)sz * SCALE; }
    void update() {
        sz = ch[0]->sz + ch[1]->sz + 1;
        for(int i = 0; i < d; ++ i) {
            maxv.x[i] = max(val.x[i],
                    max(ch[0]->maxv.x[i], ch[1]->maxv.x[i]));
            minv.x[i] = min(val.x[i],
                    min(ch[0]->minv.x[i], ch[1]->minv.x[i]));
        } }
} nodepool[N], *totnode, *null;
```

```cpp
node* newnode(poi p, int dep) {
    node *t = totnode ++; t->ch[0] = t->ch[1] = t->p = null;
    t->dep = dep; t->val = t->maxv = t->minv = p; t->sz = 1;
    return t; } // heap<pair<ll, poi>> ret; int ans_sz;
int ctr; int cmp(const poi &a,const poi &b)
{return a.x[ctr]<b.x[ctr];}
struct KDTree {
    node *root; KDTree() { root = null; }
    KDTree(poi *a, int n) { root = build(a, 0, n - 1, 0); }
    node *build(poi *a, int l, int r, int dep) {
        if (l > r) return null; ctr = dep;
        int mid = (l + r) >> 1;
        nth_element(a + l, a + mid, a + r + 1, cmp);
        node *t = newnode(a[mid], dep);
        t->set(build(a, l, mid - 1, (dep + 1) % d), 0);
        t->set(build(a, mid + 1, r, (dep + 1) % d), 1);
        t->update(); return t;
    }
    void tranv(node *t, poi *vec, int &tot) {// insert 时要
        if (t == null) return; vec[tot ++] = t->val;
        tranv(t->ch[0],vec,tot); tranv(t->ch[1],vec,tot);
    }
    void rebuild(node *t) {// insert 时要
        node *p = t->p; int tot = 0;
        tranv(t, buf, tot);
        node *u = build(buf, 0, tot - 1, t->dep);
        p->set(u, t->dir());
        for( ; p != null; p = p->p) p->update();
        if (t == root) root = u;
    }
    void insert(poi p) {// insert 时要
        if (root == null) { root = newnode(p, 0); return; }
        node *cur = root, *las = null; int dir = 0;
        for( ; cur != null; ) { las = cur;
            dir = (p.x[cur->dep] > cur->val.x[cur->dep]);
            cur = cur->ch[dir]; }
        node *t = newnode(p, (las->dep+1)%d), *bad=null;
        las->set(t, dir);
        for( ; t != null; t = t->p) {
            t->update(); if (!t->balanced()) bad = t; }
        if (bad != null) rebuild(bad);
    }
    ll calc(poi u, node *t, int d) {
        ll l = t->minv.x[d], r = t->maxv.x[d], x = u.x[d];
        if (x >= l && x <= r) return 0LL;
        ll ret = min(abs(x - l), abs(x - r));
        return ret * ret; // ret
    }
    void updateans(poi u, poi p) { /* 在这里更新答案 */ }
    void query(node *t, poi p) {
        if (t == null) return; updateans(t->val, p);
        ll eval[2] = {calc(p, t->ch[0], t->dep),
                      calc(p, t->ch[1], t->dep)};
        int cho = eval[0] <= eval[1]; // 较优侧先搜
        if(/*eval[cho^1] 可更新 ret*/)query(t->ch[cho^1],p);
        if(/*eval[cho] 可更新 ret*/)query(t->ch[cho], p);
    }
    void query(poi p) { query(root, p); }
};
void initnull(int d) { ::d = d;
    totnode = nodepool; null = totnode ++; null->sz = 0;
    for(int i = 0; i < d; ++ i) {
        null->maxv.x[i] = -INF; null->minv.x[i] = INF; }
}
```

6.4 LCT

```cpp
// 注意初始化 null 节点，单点的 is_root 初始为 true
struct Node{
    Node *ch[2], *p; int is_root, rev; bool dir();
    void set(Node*, bool); void update();
    void relax(); void app_rev();
} *null;  /* null = new Node(); */
void rot(Node *t){
    Node *p=t->p; bool d=t->dir();
    p->relax(); t->relax(); p->set(t->ch[!d],d);
    if(p->is_root) t->p=p->p,swap(p->is_root,t->is_root);
    else p->p->set(t,p->dir());
    t->set(p,!d); p->update();
}
void splay(Node *t){
    for(t->relax();!t->is_root;)
        if(t->p->is_root)rot(t);else t->dir()==t->p->dir()
            ? (rot(t->p),rot(t)) :(rot(t),rot(t));
    t->update();
}
void access(Node *t){
    for(Node *s=null; t!=null; s=t,t=t->p){
        splay(t); if (t->p == null) { /*TODO*/ }
        t->ch[1]->is_root=true; s->is_root=false;
        t->ch[1]=s; t->update();
    }
}
bool Node::dir(){ return this==p->ch[1]; }
void Node::set(Node *t,bool _d){ ch[_d]=t; t->p=this; }
void Node::update(){ }
void Node::app_rev(){ if (this == null) return;
    rev ^= true; swap(ch[0], ch[1]); }
void Node::relax() { if(this==null) return; if (rev)
    { ch[0]->app_rev(); ch[1]->app_rev(); rev = false; } }
void make_root(Node *u) {access(u);splay(u);u->app_rev();}
Node* get_root(Node *u) { access(u); splay(u);
    while (u->relax(),u->ch[0]!=null)u=u->ch[0];return u;}
void link(Node *u, Node *v) { make_root(u); u->p=v; }
void cut(Node *u, Node *v) { make_root(u); access(v);
    splay(v); v-> ch[0] = u -> p = null; u->is_root = 1; }
```

6.5 Merge-Split Treap

```cpp
// 合并两个历史版本在构造数据下深度会不断退化，可达 log 的几十倍 .
#define nil mem
struct Node {int fit; Node *l, *r; Key key; Val val, vals;
} mem[mxv] = {{0, nil, nil}}; int sz; using ptr = Node*;
#define NEW(arg...) new(mem+ ++sz)Node{rand(),nil,nil,arg}
ptr down(ptr x) {x = COPY(x); if (x->laz) {...} return x;}
pair<ptr,ptr> split(ptr x, Key key) {
    ptr t; if (x == nil) return {nil, nil}; x = down(x);
    return x->key > key // x->l->sz+1>n  key(n 个 ) 在左边
        ? (tie(t, x->l)=split(x->l, key), mp(t, renew(x)))
        : (tie(x->r, t)=split(x->r, key), mp(renew(x),t));
}
ptr merge(ptr x, ptr y) {
    if (x == nil) return y; if (y == nil) return x;
    return x->fit < y->fit // rand() % (X.sz+Y.sz) < X.sz
        ? (x = down(x), x->r = merge(x->r, y), renew(x))
        : (y = down(y), y->l = merge(x, y->l), renew(y));
}
```

6.6 Splay

```cpp
struct Node { // 注意初始化内存池和 null 节点
    int size; Node *ch[2],*p; Key key; Val val,sum,lazy;
    int dir(); void set(Node*,int); void update();
    void relax(); void app(Val);
} nodePool[MAX_NODE],*curNod,*null;
Node *newNode(Key k, Val v) { Node *t=curNod++;t->lazy=0;
```

```cpp
    t->size=1; t->key=k; t->ch[0]=t->ch[1]=t->p=null;
    t->sum=t->val=v; return t; }
struct Splay {
    Node *root;Splay(){root=newNode(INF,0); // 有两个哨兵
        root->set(newNode(-INF,0),0); root->update(); }
    void rot(Node *t) {
        Node *p=t->p;int d=t->dir();p->relax();t->relax();
        if(p==root) root=t; p->set(t->ch[!d],d);
        p->p->set(t,p->dir()); t->set(p,!d); p->update();}
    void splay(Node *t,Node *f=null) {
        for(t->relax();t->p!=f;) if(t->p->p==f) rot(t);
            else t->dir()==t->p->dir()?
                (rot(t->p),rot(t)):(rot(t),rot(t));
        t->update(); }
    Node* lower_bound(Key k) {
        Node *p=root, *res=null;
        while (p != null) {p->relax(); int d=p->key < k;
            if (!d) res = p; p=p->ch[d]; } return res;
    }
    Node* getpre(Node *x) { // x 会变成根
        splay(x); x=x->ch[0];
        while (x->relax(), x->ch[1]!=null) x=x->ch[1];
        return x; }
    Node* insert(Key k, Val v) { // 需要保证无重复 key
        Node *p=lower_bound(k); p=getpre(p); Node *t;
        p->set(t=newNode(k, v), 1); splay(p->ch[1]);
        return t; }
    void erase(Node* x) {
        splay(getpre(x), x); x->ch[0]->set(x->ch[1],1);
        (root=x->ch[0])->p=null; root->update();// 未回收
    }
    Node* kth(int k) { // 1 base
        Node *p = root; k ++; // 加上左哨兵大小 1
        while (p != null) { int ls=p->ch[0]->size;
            if (ls + 1 == k) return p; int d = ls < k;
            k -= d * (ls + 1); p=p->ch[d]; } return null;
    }
    Node *pick_by_key(Key l, Key r) { // 左闭右开
        Node *L=getpre(lower_bound(l)), *R=lower_bound(r);
        splay(R); splay(L, R); return L->ch[1];
    }
    Node *pick_by_index(int l, int r) { // 左闭右开
        Node *L=kth(l-1), *R=kth(r);
        splay(R); splay(L, R); return L->ch[1];
    }
};
void initNu(){curNod=nodePool;null=curNod++;null->size=0;}
void Node::set(Node *t,int _d){ ch[_d]=t; t->p=this; }
int Node::dir(){ return this==p->ch[1]; }
void Node::update(){ size=ch[0]->size+ch[1]->size+1;
    sum=ch[0]->sum + ch[1]->sum + val; }
void Node::relax(){
    if(lazy) ch[0]->app(lazy), ch[1]->app(lazy), lazy=0;}
void Node::app(Val c){
    if(this==null) return; lazy+=c; val+=c; sum+=c*size;}
int main() { curNod = nodePool; initNu(); }
```

7. Miscellany

7.1 日期公式

```cpp
// weekday=(id+1)%7;{Sun=0,Mon=1,...}
// getId(1, 1, 1) = 0
int getId(int y, int m, int d) {
  if (m < 3) { y --; m += 12; }
  return 365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d - 307;
}
// 当y小于0时，将除法改为向下取整后即可保证正确性，或统一加400的倍数年
auto date(int id) {
  int x=id+1789995, n, i, j, y, m, d;
  n = 4 * x / 146097; x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001; x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447; d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x; y = 100 * (n - 49) + i + x;
  return make_tuple(y, m, d); }
```

7.2 DLX - 主代码手

```cpp
struct node{
  node *left,*right,*up,*down,*col; int row,cnt;
}*head,*col[MAXC],Node[MAXNODE],*ans[MAXNODE];
int totNode;
void insert(const std::vector<int> &V,int rownum){
  std::vector<node*> N;
  for(int i=0;i<int(V.size());++i){
    node* now=Node+(totNode++); now->row=rownum;
    now->col=now->up=col[V[i]], now->down=col[V[i]]->down;
    now->up->down=now, now->down->up=now;
    now->col->cnt++; N.push_back(now);
  }
  for(int i=0;i<int(V.size());++i)
    N[i]->right=N[(i+1)%V.size()], N[i]->left=N[(i-1+V.size())%V.size()];
}
void Remove(node *x){
  x->left->right=x->right, x->right->left=x->left;
  for(node *i=x->down;i!=x;i=i->down)
    for(node *j=i->right;j!=i;j=j->right)
      j->up->down=j->down, j->down->up=j->up, --(j->col->cnt);
}
void Resume(node *x){
  for(node *i=x->up;i!=x;i=i->up)
    for(node *j=i->left;j!=i;j=j->left)
      j->up->down=j->down->up=j, ++(j->col->cnt);
  x->left->right=x, x->right->left=x;
}
bool search(int tot){
  if(head->right==head) return true;
  node *choose=NULL;
  for(node *i=head->right;i!=head;i=i->right){
    if(choose==NULL||choose->cnt>i->cnt) choose=i;
    if(choose->cnt<2) break;
  }
  Remove(choose);
  for(node *i=choose->down;i!=choose;i=i->down){
    for(node *j=i->right;j!=i;j=j->right) Remove(j->col);
    ans[tot]=i;
    if(search(tot+1)) return true;
    ans[tot]=NULL;
    for(node *j=i->left;j!=i;j=j->left) Resume(j->col);
  }
  Resume(choose);
  return false;
}
void prepare(int totC){
  head=Node+totC;
  for(int i=0;i<totC;++i) col[i]=Node+i;
  totNode=totC+1;
  for(int i=0;i<=totC;++i){
    (Node+i)->right=Node+(i+1)%(totC+1);
    (Node+i)->left=Node+(i+totC)%(totC+1);
    (Node+i)->up=(Node+i)->down=Node+i;
  }
}
```

7.3 直线下格点统计

```cpp
// \sum_{i=0}^{n-1} \lfloor \frac{a+bi}{m} \rfloor, n,m,a,b > 0
ll solve(ll n, ll a, ll b, ll m){
  if (b == 0) return n * (a / m);
```

```java
  if (a >= m) return n * (a / m) + solve(n, a % m, b, m);
  if (b >= m) return (n - 1) * n / 2 * (b / m) + solve(n, a, b % m, m);
  return solve((a + b * n) / m, (a + b * n) % m, m, b);
}
```

8. Others

8.1 Java Template

```java
import java.io.*; import java.util.*; import java.math.*;
public class Main {
static class solver { public void run(Scanner cin, PrintStream out) {} }
public static void main(String[] args) {
// Fast Reader & Big Numbers
InputReader in = new InputReader(System.in);
PrintWriter out = new PrintWriter(System.out);
BigInteger c = in.nextInt();
out.println(c.toString(8)); out.close(); // as Oct
BigDecimal d = new BigDecimal(10.0);
// d=d.divide(num, length, BigDecimal.ROUND_HALF_UP)
d.setScale(2, BigDecimal.ROUND_FLOOR); // 用于输出
System.out.printf("%.6f\n", 1.23); // C 格式
BigInteger num = BigInteger.valueOf(6);
num.isProbablePrime(10); // 1 - 1 / 2 ^ certainty
BigInteger rev = num.modPow(BigInteger.valueOf(-1), BigInteger.valueOf(25));  // rev=6^-1mod25 要互质
num = num.nextProbablePrime(); num.intValue();
Scanner cin=new Scanner(System.in);//SimpleReader
int n = cin.nextInt();
int [] a = new int [n]; // 初值未定义
// Random rand.nextInt(N) [0,N)
Arrays.sort(a, 5, 10, cmp); // sort(a+5, a+10)
ArrayList<Long> list = new ArrayList(); // vector
// .add(val) .add(pos, val) .remove(pos)
Comparator cmp=new Comparator<Long>(){ // 自定义逆序
  @Override public int compare(Long o1, Long o2) {
  /* o1 < o2 ? 1 :( o1 > o2  ? -1 : 0) */ } };
// Collections. shuffle(list) sort(list, cmp)
Long [] tmp = list.toArray(new Long [0]);
// list.get(pos) list.size()
Map<Integer,String> m = new HashMap<Integer,String>();
//m.put(key,val) get(key) containsKey(key) remove(key)
for (Map.Entry<Integer,String> entry:m.entrySet()); //entry.getKey() getValue()
Set<String> s = new HashSet<String>(); // TreeSet
//s.add(val)contains(val)remove(val);for(var : s)
solver Task=new solver();Task.run(cin,System.out);
PriorityQueue<Integer> Q=new PriorityQueue<Integer>();
// Q. offer(val) poll() peek() size()
// Write to file : FileWriter
} static class InputReader { // Fast Reader
public BufferedReader reader;
public StringTokenizer tokenizer;
public InputReader(InputStream stream) {
    reader = new BufferedReader(new InputStreamReader(stream), 32768);
    tokenizer = null; }
public String next() {
    while (tokenizer == null || !tokenizer.hasMoreTokens()) {
        try { String line = reader.readLine();
            /*line == null ? end of file*/
            tokenizer = new StringTokenizer(line);
        } catch (IOException e) {
            throw new RuntimeException(e); }
    } return tokenizer.nextToken(); }
public BigInteger nextInt() {
    //return Long.parseLong(next()); Double Integer
    return new BigInteger(next(), 16); // as Hex
} } }
```

8.2 Formulas

8.2.1 Arithmetic Function

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

$$J_k(n) = n^k \prod_{p|n}(1-\frac{1}{p^k})$$

$J_k(n)$ is the number of k-tuples of positive integers all less than or equal to n that form a coprime $(k+1)$-tuple together with n.

$$\sum_{\delta|n} J_k(\delta) = n^k$$

$$\sum_{\delta|n} \delta^s J_r(\delta) J_s(\frac{n}{\delta}) = J_{r+s}(n)$$

$$\sum_{\delta|n} \varphi(\delta) d(\frac{n}{\delta}) = \sigma(n), \quad \sum_{\delta|n} |\mu(\delta)| = 2^{\omega(n)}$$

$$\sum_{\delta|n} 2^{\omega(\delta)} = d(n^2), \quad \sum_{\delta|n} d(\delta^2) = d^2(n)$$

$$\sum_{\delta|n} d(\frac{n}{\delta}) 2^{\omega(\delta)} = d^2(n), \quad \sum_{\delta|n} \frac{\mu(\delta)}{\delta} = \frac{\varphi(n)}{n}$$

$$\sum_{\delta|n} \frac{\mu(\delta)}{\varphi(\delta)} = d(n), \quad \sum_{\delta|n} \frac{\mu^2(\delta)}{\varphi(\delta)} = \frac{n}{\varphi(n)}$$

$$n | \varphi(a^n - 1)$$

$$\sum_{\substack{1\le k\le n \\ \gcd(k,n)=1}} f(\gcd(k-1,n)) = \varphi(n) \sum_{d|n} \frac{(\mu * f)(d)}{\varphi(d)}$$

$$\varphi(\operatorname{lcm}(m,n))\varphi(\gcd(m,n)) = \varphi(m)\varphi(n)$$

$$\sum_{\delta|n} d^3(\delta) = (\sum_{\delta|n} d(\delta))^2$$

$$d(uv) = \sum_{\delta|\gcd(u,v)} \mu(\delta) d(\frac{u}{\delta}) d(\frac{v}{\delta})$$

$$\sigma_k(u)\sigma_k(v) = \sum_{\delta|\gcd(u,v)} \delta^k \sigma_k(\frac{uv}{\delta^2})$$

$$\mu(n) = \sum_{k=1}^{n} [\gcd(k,n)=1] \cos 2\pi\frac{k}{n}$$

$$\varphi(n) = \sum_{k=1}^{n} [\gcd(k,n)=1] = \sum_{k=1}^{n} \gcd(k,n) \cos 2\pi\frac{k}{n}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n} (f*g)(k) \\ \sum_{k=1}^{n} S(\lfloor \frac{n}{k} \rfloor) = \sum_{i=1}^{n} f(i) \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} (g*1)(j) \end{cases}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n} (f\cdot g)(k), g \text{ completely multiplicative} \\ \sum_{k=1}^{n} S(\lfloor \frac{n}{k} \rfloor) g(k) = \sum_{k=1}^{n} (f*1)(k) g(k) \end{cases}$$

8.2.2 Binomial Coefficients

$$\binom{n}{k} = (-1)^k \binom{k-n-1}{k}$$

$$\sum_{k\le n}\binom{r+k}{k}=\binom{r+n+1}{n}$$
$$\sum_{k=0}^{n}\binom{k}{m}=\binom{n+1}{m+1}$$
$$\sqrt{1+z}=1+\sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{k\times 2^{2k-1}}\binom{2k-2}{k-1}z^k$$
$$\sum_{k=0}^{r}\binom{r-k}{m}\binom{s+k}{n}=\binom{r+s+1}{m+n+1}$$
$$C_{n,m}=\binom{n+m}{m}-\binom{n+m}{m-1},n\ge m$$
$$\binom{n}{k}\equiv[n\&k=k]\pmod 2$$

8.2.3 Fibonacci Numbers

$$F(z)=\frac{z}{1-z-z^2}$$
$$f_n=\frac{\phi^n-\hat{\phi}^n}{\sqrt{5}},\phi=\frac{1+\sqrt{5}}{2},\hat{\phi}=\frac{1-\sqrt{5}}{2}$$
$$\sum_{k=1}^{n}f_k=f_{n+2}-1$$
$$\sum_{k=1}^{n}f_k^2=f_nf_{n+1}$$
$$\sum_{k=0}^{n}f_kf_{n-k}=\frac{1}{5}(n-1)f_n+\frac{2}{5}nf_{n-1}$$
$$f_n^2+(-1)^n=f_{n+1}f_{n-1}$$
$$f_{n+k}=f_nf_{k+1}+f_{n-1}f_k$$
$$f_{2n+1}=f_n^2+f_{n+1}^2$$
$$(-1)^kf_{n-k}=f_nf_{k-1}-f_{n-1}f_k$$
$$\text{Modulo } f_n, f_{mn+r}\equiv\begin{cases}f_r, & m \bmod 4=0;\\ (-1)^{r+1}f_{n-r}, & m \bmod 4=1;\\ (-1)^nf_r, & m \bmod 4=2;\\ (-1)^{r+1+n}f_{n-r}, & m \bmod 4=3.\end{cases}$$

8.2.4 Stirling Cycle Numbers

$$\begin{bmatrix}n+1\\k\end{bmatrix}=n\begin{bmatrix}n\\k\end{bmatrix}+\begin{bmatrix}n\\k-1\end{bmatrix},\ \begin{bmatrix}n+1\\2\end{bmatrix}=n!H_n$$
$$x^{\underline{n}}=\sum_k\begin{bmatrix}n\\k\end{bmatrix}(-1)^{n-k}x^k,\ x^{\overline{n}}=\sum_k\begin{bmatrix}n\\k\end{bmatrix}x^k$$

8.2.5 Stirling Subset Numbers

$$\begin{Bmatrix}n+1\\k\end{Bmatrix}=k\begin{Bmatrix}n\\k\end{Bmatrix}+\begin{Bmatrix}n\\k-1\end{Bmatrix}$$
$$x^n=\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}x^{\underline{k}}=\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}(-1)^{n-k}x^{\overline{k}}$$
$$m!\begin{Bmatrix}n\\m\end{Bmatrix}=\sum_k\binom{m}{k}k^n(-1)^{m-k}$$

8.2.6 Eulerian Numbers

$$\left\langle{n\atop k}\right\rangle=(k+1)\left\langle{n-1\atop k}\right\rangle+(n-k)\left\langle{n-1\atop k-1}\right\rangle$$
$$x^n=\sum_k\left\langle{n\atop k}\right\rangle\binom{x+k}{n}$$
$$\left\langle{n\atop m}\right\rangle=\sum_{k=0}^{m}\binom{n+1}{k}(m+1-k)^n(-1)^k$$

8.2.7 Harmonic Numbers

$$\sum_{k=1}^{n}H_k=(n+1)H_n-n$$
$$\sum_{k=1}^{n}kH_k=\frac{n(n+1)}{2}H_n-\frac{n(n-1)}{4}$$
$$\sum_{k=1}^{n}\binom{k}{m}H_k=\binom{n+1}{m+1}(H_{n+1}-\frac{1}{m+1})$$

8.2.8 Pentagonal Number Theorem

$$\prod_{n=1}^{\infty}(1-x^n)=\sum_{n=-\infty}^{\infty}(-1)^kx^{k(3k-1)/2}$$
$$p(n)=p(n-1)+p(n-2)-p(n-5)-p(n-7)+\cdots$$
$$f(n,k)=p(n)-p(n-k)-p(n-2k)+p(n-5k)+p(n-7k)-\cdots$$

8.2.9 Bell Numbers

$$B_{n+1}=\sum_{k=0}^{n}\binom{n}{k}B_k$$
$$B_{p^m+n}\equiv mB_n+B_{n+1}\pmod p$$

8.2.10 Bernoulli Numbers

$$B_n=1-\sum_{k=0}^{n-1}\binom{n}{k}\frac{B_k}{n-k+1}$$
$$G(x)=\sum_{k=0}^{\infty}\frac{B_k}{k!}x^k=\frac{1}{\sum_{k=0}^{\infty}\frac{x^k}{(k+1)!}}$$
$$S_m(n)=\frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_kn^{m-k+1}$$

8.2.11 Tetrahedron Volume

$$V=\frac{\sqrt{4u^2v^2w^2-\sum_{cyc}u^2(v^2+w^2-U^2)^2+\prod_{cyc}(v^2+w^2-U^2)}}{12}$$

8.2.12 BEST Thoerem

Counting the number of different Eulerian circuits in directed graphs.

$$\mathrm{ec}(G)=t_w(G)\prod_{v\in V}(\deg(v)-1)!$$

When calculating $t_w(G)$ for directed multigraphs, the entry $q_{i,j}$ for distinct i and j equals $-m$, where m is the number of edges from i to j, and the entry $q_{i,i}$ equals the indegree of i minus the number of loops at i. It is a property of Eulerian graphs that $\mathrm{tv}(G)=\mathrm{tw}(G)$ for every two vertices v and w in a connected Eulerian graph G.

8.2.13 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 $\frac{4r\sin^3(\theta/2)}{3(\theta-\sin(\theta))}$

8.2.14 Others

$$S_j=\sum_{k=1}^{n}x_k^j$$
$$h_m=\sum_{1\le j_1<\cdots<j_m\le n}x_{j_1}\cdots x_{j_m}$$
$$H_m=\sum_{1\le j_1\le\cdots\le j_m\le n}x_{j_1}\cdots x_{j_m}$$

$$h_n = \frac{1}{n}\sum_{k=1}^{n}(-1)^{k+1}S_k h_{n-k}$$

$$H_n = \frac{1}{n}\sum_{k=1}^{n}S_k H_{n-k}$$

$$\sum_{k=0}^{n} kc^k = \frac{nc^{n+2}-(n+1)c^{n+1}+c}{(c-1)^2}$$

$$n! = \sqrt{2\pi n}(\frac{n}{e})^n(1+\frac{1}{12n}+\frac{1}{288n^2}+O(\frac{1}{n^3}))$$

$$\max\{x_a-x_b, y_a-y_b, z_a-z_b\} - \min\{x_a-x_b, y_a-y_b, z_a-z_b\} = \frac{1}{2}\sum_{cyc}|(x_a-y_a)-(x_b-y_b)|$$

$$(a+b)(b+c)(c+a) = \frac{(a+b+c)^3-a^3-b^3-c^3}{3}$$

8.3 Integration Table

8.3.1 $ax^2+bx+c(a>0)$

1. $\int \frac{\mathrm{d}x}{ax^2+bx+c} = \begin{cases} \frac{2}{\sqrt{4ac-b^2}}\arctan\frac{2ax+b}{\sqrt{4ac-b^2}}+C & (b^2<4ac) \\ \frac{1}{\sqrt{b^2-4ac}}\ln\left|\frac{2ax+b-\sqrt{b^2-4ac}}{2ax+b+\sqrt{b^2-4ac}}\right|+C & (b^2>4ac) \end{cases}$
2. $\int \frac{x}{ax^2+bx+c}\mathrm{d}x = \frac{1}{2a}\ln|ax^2+bx+c| - \frac{b}{2a}\int\frac{\mathrm{d}x}{ax^2+bx+c}$

8.3.2 $\sqrt{\pm ax^2+bx+c}(a>0)$

1. $\int\frac{\mathrm{d}x}{\sqrt{ax^2+bx+c}} = \frac{1}{\sqrt{a}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}|+C$
2. $\int\sqrt{ax^2+bx+c}\mathrm{d}x = \frac{2ax+b}{4a}\sqrt{ax^2+bx+c} + \frac{4ac-b^2}{8\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}|+C$
3. $\int\frac{x}{\sqrt{ax^2+bx+c}}\mathrm{d}x = \frac{1}{a}\sqrt{ax^2+bx+c} - \frac{b}{2\sqrt{a^3}}\ln|2ax+b+2\sqrt{a}\sqrt{ax^2+bx+c}|+C$
4. $\int\frac{\mathrm{d}x}{\sqrt{c+bx-ax^2}} = -\frac{1}{\sqrt{a}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}+C$
5. $\int\sqrt{c+bx-ax^2}\mathrm{d}x = \frac{2ax-b}{4a}\sqrt{c+bx-ax^2} + \frac{b^2+4ac}{8\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}+C$
6. $\int\frac{x}{\sqrt{c+bx-ax^2}}\mathrm{d}x = -\frac{1}{a}\sqrt{c+bx-ax^2}+\frac{b}{2\sqrt{a^3}}\arcsin\frac{2ax-b}{\sqrt{b^2+4ac}}+C$

8.3.3 $\sqrt{\pm\frac{x-a}{x-b}}$或$\sqrt{(x-a)(x-b)}$

1. $\int\frac{\mathrm{d}x}{\sqrt{(x-a)(b-x)}} = 2\arcsin\sqrt{\frac{x-a}{b-x}}+C\ (a<b)$
2.

$$\int\sqrt{(x-a)(b-x)}\mathrm{d}x = \frac{2x-a-b}{4}\sqrt{(x-a)(b-x)}+ \frac{(b-a)^2}{4}\arcsin\sqrt{\frac{x-a}{b-x}}+C,(a<b) \quad (10)$$

8.3.4 三角函数的积分

1. $\int\tan x\mathrm{d}x = -\ln|\cos x|+C$
2. $\int\cot x\mathrm{d}x = \ln|\sin x|+C$
3. $\int\sec x\mathrm{d}x = \ln\left|\tan\left(\frac{\pi}{4}+\frac{x}{2}\right)\right|+C = \ln|\sec x+\tan x|+C$
4. $\int\csc x\mathrm{d}x = \ln\left|\tan\frac{x}{2}\right|+C = \ln|\csc x-\cot x|+C$
5. $\int\sec^2 x\mathrm{d}x = \tan x+C$
6. $\int\csc^2 x\mathrm{d}x = -\cot x+C$
7. $\int\sec x\tan x\mathrm{d}x = \sec x+C$
8. $\int\csc x\cot x\mathrm{d}x = -\csc x+C$
9. $\int\sin^2 x\mathrm{d}x = \frac{x}{2}-\frac{1}{4}\sin 2x+C$
10. $\int\cos^2 x\mathrm{d}x = \frac{x}{2}+\frac{1}{4}\sin 2x+C$
11. $\int\sin^n x\mathrm{d}x = -\frac{1}{n}\sin^{n-1}x\cos x+\frac{n-1}{n}\int\sin^{n-2}x\mathrm{d}x$
12. $\int\cos^n x\mathrm{d}x = \frac{1}{n}\cos^{n-1}x\sin x+\frac{n-1}{n}\int\cos^{n-2}x\mathrm{d}x$
13. $\int\frac{\mathrm{d}x}{\sin^n x} = -\frac{1}{n-1}\frac{\cos x}{\sin^{n-1}x}+\frac{n-2}{n-1}\int\frac{\mathrm{d}x}{\sin^{n-2}x}$
14. $\int\frac{\mathrm{d}x}{\cos^n x} = \frac{1}{n-1}\frac{\sin x}{\cos^{n-1}x}+\frac{n-2}{n-1}\int\frac{\mathrm{d}x}{\cos^{n-2}x}$
15.

$$\begin{aligned}&\int\cos^m x\sin^n x\mathrm{d}x\\ =&\frac{1}{m+n}\cos^{m-1}x\sin^{n+1}x+\frac{m-1}{m+n}\int\cos^{m-2}x\sin^n x\mathrm{d}x\\ =&-\frac{1}{m+n}\cos^{m+1}x\sin^{n-1}x+\frac{n-1}{m+1}\int\cos^m x\sin^{n-2}x\mathrm{d}x\end{aligned}$$

16. $\int\frac{\mathrm{d}x}{a+b\sin x} = \begin{cases}\frac{2}{\sqrt{a^2-b^2}}\arctan\frac{a\tan\frac{x}{2}+b}{\sqrt{a^2-b^2}}+C & (a^2>b^2)\\ \frac{1}{\sqrt{b^2-a^2}}\ln\left|\frac{a\tan\frac{x}{2}+b-\sqrt{b^2-a^2}}{a\tan\frac{x}{2}+b+\sqrt{b^2-a^2}}\right|+C & (a^2<b^2)\end{cases}$
17. $\int\frac{\mathrm{d}x}{a+b\cos x} = \begin{cases}\frac{2}{a+b}\sqrt{\frac{a+b}{a-b}}\arctan\left(\sqrt{\frac{a-b}{a+b}}\tan\frac{x}{2}\right)+C & (a^2>b^2)\\ \frac{1}{a+b}\sqrt{\frac{a+b}{a-b}}\ln\left|\frac{\tan\frac{x}{2}+\sqrt{\frac{a+b}{b-a}}}{\tan\frac{x}{2}-\sqrt{\frac{a+b}{b-a}}}\right|+C & (a^2<b^2)\end{cases}$
18. $\int\frac{\mathrm{d}x}{a^2\cos^2 x+b^2\sin^2 x} = \frac{1}{ab}\arctan\left(\frac{b}{a}\tan x\right)+C$
19. $\int\frac{\mathrm{d}x}{a^2\cos^2 x-b^2\sin^2 x} = \frac{1}{2ab}\ln\left|\frac{b\tan x+a}{b\tan x-a}\right|+C$
20. $\int x\sin ax\mathrm{d}x = \frac{1}{a^2}\sin ax-\frac{1}{a}x\cos ax+C$
21. $\int x^2\sin ax\mathrm{d}x = -\frac{1}{a}x^2\cos ax+\frac{2}{a^2}x\sin ax+\frac{2}{a^3}\cos ax+C$
22. $\int x\cos ax\mathrm{d}x = \frac{1}{a^2}\cos ax+\frac{1}{a}x\sin ax+C$
23. $\int x^2\cos ax\mathrm{d}x = \frac{1}{a}x^2\sin ax+\frac{2}{a^2}x\cos ax-\frac{2}{a^3}\sin ax+C$

8.3.5 反三角函数的积分(其中 $a>0$)

1. $\int\arcsin\frac{x}{a}\mathrm{d}x = x\arcsin\frac{x}{a}+\sqrt{a^2-x^2}+C$
2. $\int x\arcsin\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arcsin\frac{x}{a}+\frac{x}{4}\sqrt{x^2-x^2}+C$
3. $\int x^2\arcsin\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arcsin\frac{x}{a}+\frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2}+C$
4. $\int\arccos\frac{x}{a}\mathrm{d}x = x\ arccos\frac{x}{a}-\sqrt{a^2-x^2}+C$
5. $\int x\arccos\frac{x}{a}\mathrm{d}x = (\frac{x^2}{2}-\frac{a^2}{4})\arccos\frac{x}{a}-\frac{x}{4}\sqrt{a^2-x^2}+C$
6. $\int x^2\arccos\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arccos\frac{x}{a}-\frac{1}{9}(x^2+2a^2)\sqrt{a^2-x^2}+C$
7. $\int\arctan\frac{x}{a}\mathrm{d}x = x\arctan\frac{x}{a}-\frac{a}{2}\ln(a^2+x^2)+C$
8. $\int x\arctan\frac{x}{a}\mathrm{d}x = \frac{1}{2}(a^2+x^2)\arctan\frac{x}{a}-\frac{a}{2}x+C$
9. $\int x^2\arctan\frac{x}{a}\mathrm{d}x = \frac{x^3}{3}\arctan\frac{x}{a}-\frac{a}{6}x^2+\frac{a^3}{6}\ln(a^2+x^2)+C$

8.3.6 指数函数的积分

1. $\int a^x\mathrm{d}x = \frac{1}{\ln a}a^x+C$
2. $\int \mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}a^{ax}+C$
3. $\int x\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a^2}(ax-1)a^{ax}+C$
4. $\int x^n\mathrm{e}^{ax}\mathrm{d}x = \frac{1}{a}x^n\mathrm{e}^{ax}-\frac{n}{a}\int x^{n-1}\mathrm{e}^{ax}\mathrm{d}x$
5. $\int xa^x\mathrm{d}x = \frac{x}{\ln a}a^x-\frac{1}{(\ln a)^2}a^x+C$
6. $\int x^n a^x\mathrm{d}x = \frac{1}{\ln a}x^n a^x-\frac{n}{\ln a}\int x^{n-1}a^x\mathrm{d}x$
7. $\int\mathrm{e}^{ax}\sin bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(a\sin bx-b\cos bx)+C$
8. $\int\mathrm{e}^{ax}\cos bx\mathrm{d}x = \frac{1}{a^2+b^2}\mathrm{e}^{ax}(b\sin bx+a\cos bx)+C$
9. $\int\mathrm{e}^{ax}\sin^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\sin^{n-1}bx(a\sin bx - nb\cos bx)+\frac{n(n-1)b^2}{a^2+b^2n^2}\int\mathrm{e}^{ax}\sin^{n-2}bx\mathrm{d}x$
10. $\int\mathrm{e}^{ax}\cos^n bx\mathrm{d}x = \frac{1}{a^2+b^2n^2}\mathrm{e}^{ax}\cos^{n-1}bx(a\cos bx + nb\sin bx)+\frac{n(n-1)b^2}{a^2+b^2n^2}\int\mathrm{e}^{ax}\cos^{n-2}bx\mathrm{d}x$

8.3.7 对数函数的积分

1. $\int\ln x\mathrm{d}x = x\ln x-x+C$
2. $\int\frac{\mathrm{d}x}{x\ln x} = \ln\big|\ln x\big|+C$
3. $\int x^n\ln x\mathrm{d}x = \frac{1}{n+1}x^{n+1}(\ln x-\frac{1}{n+1})+C$
4. $\int(\ln x)^n\mathrm{d}x = x(\ln x)^n-n\int(\ln x)^{n-1}\mathrm{d}x$
5. $\int x^m(\ln x)^n\mathrm{d}x = \frac{1}{m+1}x^{m+1}(\ln x)^n-\frac{n}{m+1}\int x^m(\ln x)^{n-1}\mathrm{d}x$