算法模板

刘阳 袁昊 邓宏亮

2019 年 9 月 3 日

目录

1 计算几何

1.1 自适应辛普森

```
typedef double db;
struct Simpson {
  /* 系数 */
  db F(db x) { return /* 表达式 */; }
  db Simpson(db l, db r) {
    db m = (l + r) / 2.0;
    return (F(l) + 4 * F(m) + F(r)) * (r - l) / 6.0;
  }
  db Asr(db l, db r, db ans, db eps) {
    db m = (l + r) / 2.0;
    db l_ans = Simpson(l, m), r_ans = Simpson(m, r);
    if (fabs(l_ans + r_ans - ans) <= 15.0 * eps) return
         l_ans + r_ans + (l_ans + r_ans - ans) / 15.0;
    return Asr(l, m, l_ans, eps / 2.0) + Asr(m, r, r_ans,
          eps / 2.0);
  }
};
```

1.2 立体几何

```
typedef double db;
const db inf = 1e100;
const db eps = 1e-9;
const db pi = acos(-1.);
const db delta = 0.98;
int Sgn(db k) { return std::fabs(k) < eps ? 0 : (k < 0 ?
     -1 : 1); }
int Cmp(db k1, db k2) { return Sgn(k1 - k2); }
db Min(db k1, db k2) { return Cmp(k1, k2) < 0 ? k1 : k2;
     }
db Max(db k1, db k2) { return Cmp(k1, k2) > 0 ? k1 : k2;
     }
struct Point { db x, y, z; }
bool operator == (Point k1, Point k2) { return !Sgn(k1.x
     - k2.x) && !Sgn(k1.y, k2.y) && !Sgn(k1.z, k2.z); }
Point operator + (Point k1, Point k2) { return (Point){k1
     .x + k2.x, k1.y + k2.y, k1.z + k2.z}; }
Point operator - (Point k1, Point k2) { return (Point){k1
     .x - k2.x, k1.y - k2.y, k1.z - k2.z};
Point operator * (Point k1, db k2) { return (Point){k1.x
     * k2, k1.y * k2, k1.z * k2};  }
Point operator / (Point k1, db k2) { return (Point){k1.x
     / k2, k1.y / k2, k1.z / k2};  }
db operator * (Point k1, Point k2) { k1.x * k2.x + k1.y *
      k2.y + k1.z * k2.z; }
Point operator ^ (Point k1, Point k2) { return (Point){k1
     .y * k2.z - k1.z * k2.y, k1.z * k2.x - k1.x * k2.z,
     k1.x * k2.y - k1.y * k2.x}; }
db GetLen(Point k) { return std::sqrt(k * k); }
db GetLen2(Point k) { return k * k; }
Point GetUnit(Point k) { return k / GetLen(k); }
db GetDis(Point k1, Point k2) { return GetLen(k2 - k1); }
db GetDis2(Point k1, Point k2) { return GetLen2(k2 - k1);
      }
db GetMinSphereR(std::vector<Point> p) {
  Point cur = p[0];
  db pro = 10000, ret = inf;
  while (pro > eps) {
    int idx = 0;
    for (int i = 0; i < p.size(); ++i)
      if (Cmp(GetDis(cur, p[i]), GetDis(cur, p[idx])) >
           0)
        idx = i;
    db r = GetDis(cur, p[idx]);
    ret = Min(ret, r);
    cur = cur + (p[idx] - cur) / r * pro;
    pro *= delta;
  }
  return ret;
}
struct Line { Point s, t; };
struct Seg: public Line {};
db GetLen(Seg k) { return GetDis(k.s, k.t); }
db GetLen2(Seg k) { return GetDis2(k.s, k.t); }
db GetDis(Point k1, Line k2) { return std::fabs((k1 - k2.
     s) ^ (k2.t - k2.s)) / GetLen(k2); }
db GetDis(Point k1, Seg k2) {
  if (Sgn((k1 - k2.s) * (k2.t - k2.s)) < 0 || Sgn((k1 -
       k2.t) * (k2.s - k2.t)) < 0)
    return Min(GetDis(k1, k2.s), GetDis(k1, k2.t));
  return GetDis(k1, Seg); // Point to Line dis
}
struct Sphere { Point o; db r; };
db GetV(Sphere k) { return 4. / 3. * pi * k.r * k.r * k.r
     ; }
db GetInterV(Sphere k1, Sphere k2) {
  db dis = GetDisP2P(k1.o, k2.o);
  if (Sgn(dis - k1.r - k2.r) >= 0) return ret;
  if (Sgn(k2.r - (dis + k1.r)) >= 0) return GetV(k1);
  else if (Sgn(k1.r - (dis + k2.r)) >= 0) return GetV(k2)
       ;
  db len1 = ((k1.r * k1.r - k2.r * k2.r) / dis + dis) /
       2;
  db len2 = dis - len1;
  db x1 = k1.r - len1, x2 = k2.r - len2;
  db v1 = pi * x1 * x1 * (k1.r - x1 / 3.0);
  db v2 = pi * x2 * x2 * (k2.r - x2 / 3.0);
  return v1 + v2;
}
```

1.3 皮克定理

```
polygon: S = in + (on / 2) - 1
```

1.4 平面几何

```
typedef double db; // typedef long double db;
const db inf = 1e100;
const db eps = 1e-9;
const db delta = 0.98;
int Sgn(db k) { return std::fabs(k) < eps ? 0 : (k < 0 ? -1 : 1); }
int Cmp(db k1, db k2) { return Sgn(k1 - k2); }
bool IsInMid(db k1, db k2, db k3) { return Sgn(k1 - k3) * Sgn(k2 - k3) <= 0; }
db Max(db k1, db k2) { return Cmp(k1, k2) > 0 ? k1 : k2; }
db Min(db k1, db k2) { return Cmp(k1, k2) < 0 ? k1 : k2; }
struct Point { db x, y; };
bool operator == (Point k1, Point k2) { return !Cmp(k1.x, k2.x) && !Cmp(k1.y, k2.y); }
Point operator + (Point k1, Point k2) { return (Point){k1.x + k2.x, k1.y + k2.y}; }
Point operator - (Point k1, Point k2) { return (Point){k1.x - k2.x, k1.y - k2.y}; }
Point operator * (Point k1, db k2) { return (Point){k1.x * k2, k1.y * k2}; }
Point operator / (Point k1, db k2) { return (Point){k1.x / k2, k1.y / k2}; }
db operator * (Point k1, Point k2) { return k1.x * k2.x + k1.y * k2.y; }
db operator ^ (Point k1, Point k2) { return k1.x * k2.y - k1.y * k2.x; }
bool IsInMid(Point k1, Point k2, Point k3) { return IsInMid(k1.x, k2.x, k3.x) && IsInMid(k1.y, k2.y, k3.y); }
db GetLen(Point k) { return std::sqrt(k * k); }
db GetLen2(Point k) { return k * k; }
Point GetUnit(Point k) { return k / GetLen(k); }
db GetDis(Point k1, Point k2) { return GetLen(k2 - k1); }
db GetDis2(Point k1, Point k2) { return GetLen2(k2 - k1); }
db GetAng(Point k1, Point k2) { return std::atan2((k1 ^ k2), (k1 * k2)); }
Point Rotate(Point k, db ang) { return (Point){k.x * std::cos(ang) - k.y * std::sin(ang), k.x * std::sin(ang) + k.y + std::cos(ang)}; }
Point Rotate90(Point k) { return (Point){-k.y, k.x}; }
struct Line { Point s, t; };
struct Seg: public Line {}; // typedef Line Seg
db GetLen(Seg k) { return GetDis(k.s, k.t); }
bool IsOn(Point k1, Seg k2) { return !Sgn((k1 - k2.s) ^ (k2.t - k2.s)) && Sgn((k1 - k2.s) * (k1 - k2.t)) <= 0; }
Point Proj(Point k1, Line k2) { Point k = k2.t - k2.s; return k2.s + k * (((k1 - k2.s) * k) / GetLen(k)); }
Point Reflect(Point k1, Line k2) { return Proj(k1, k2) * 2 - k1; }
bool IsParallel(Line k1, Line k2) { return !Sgn((k1.s - k1.t) ^ (k2.s - k2.t)); }
bool IsInter(Seg k1, Seg k2) {
  return
    Cmp(Max(k1.s.x, k1.t.x), Min(k2.s.x, k2.t.x)) >= 0 &&
    Cmp(Max(k2.s.x, k2.t.x), Min(k1.s.x, k1.t.x)) >= 0 &&
    Cmp(Max(k1.s.y, k1.t.y), Min(k2.s.y, k2.t.y)) >= 0 &&
    Cmp(Max(k2.s.y, k2.t.y), Min(k1.s.y, k1.t.y)) >= 0 &&
    Sgn((k2.s - k1.t) ^ (k1.s - k1.t)) * Sgn((k2.t - k1.t) ^ (k1.s - k1.t)) <= 0 &&
    Sgn((k1.s - k2.t) ^ (k2.s - k2.t)) * Sgn((k1.t - k2.t) ^ (k2.s - k2.t)) <= 0;
}
bool IsInter(Line k1, Seg k2) {
  return Sgn((k2.s - k1.t) ^ (k1.s - k1.t)) * Sgn((k2.t - k1.t) ^ (k1.s - k1.t)) <= 0;
}
bool IsInter(Line k1, Line k2) {
  if (!IsParallel(k1, k2)) return true;
  return !Sgn((k1.s - k2.s) ^ (k2.t - k2.s));
}
db GetDis(Point k1, Line k2) {
  return std::fabs((k1 - k2.s) ^ (k2.t - k2.s)) / GetLen(k2);
}
db GetDis(Point k1, Seg k2) {
  if (Sgn((k1 - k2.s) * (k2.t - k2.s)) < 0 || Sgn((k1 - k2.t) * (k2.s - k2.t)) < 0) return Min(GetDis(k1, k2.s), GetDis(k1, k2.t));
  return GetDis(k1, k2);
}
db GetDis(Seg k1, Seg k2) {
  if (IsInter(k1, k2)) return 0.;
  else return Min(Min(GetDis(k1.s, k2), GetDis(k1.t, k2)), Min(GetDis(k1, k2.s), GetDis(k1, k2.t)));
}
Point Cross(Line k1, Line k2) {
  db w1 = (k1.s - k2.s) ^ (k2.t - k2.s), w2 = (k2.t - k2.s) ^ (k1.t - k2.s);
  return (k1.s * w2 + k1.t * w1) / (w1 + w2);
}
// 平面直线图(PSLG)
struct Edge { int u, v; db ang; };
```

```
struct PSLG {
  int n, m, face_cnt; // 多边形数
  Point p[maxn];
  std::vector<Edge> e;
  std::vector<int> g[maxn];
  bool vis[maxn * 2];
  int left[maxn * 2], prev[maxn * 2];
  std::vector<Polygon> faces; // 多边形
  db area[maxn]; // 多边形面积
  void Init() {
    n = m = 0;
    for (int i = 0; i < n; ++i) g[i].clear();
    e.clear();
    faces.clear();
  }
  // 有向线段pt.x->pt.y的极角
  db GetAng(Point pt) {
    return std::atan2(pt.y, pt.x);
  }
  void AddEdge(int u, int v) {
    e.push_back((Edge){u, v, GetAng(p[v] - p[u])});
    e.push_back((Edge){v, u, GetAng(p[u] - p[v])});
    m = e.size();
    g[u].push_back(m - 2);
    g[v].push_back(m - 1);
  }
  // 找出faces并计算面积
  void Build() {
    for (int u = 0; u < n; ++u) {
      int sz = g[u].size();
      for (int i = 0; i < sz; ++i)
        for (int j = i + 1; j < sz; ++j)
          if (e[g[u][i]].ang > e[g[u][j]].ang) std::swap(
              g[u][i], g[u][j]);
      for (int i = 0; i < sz; ++i) prev[g[u][(i + 1) % sz
          ]] = g[u][i];
    }
    face_cnt = 0;
    memset(vis, false, sizeof(vis));
    for (int u = 0; u < n; ++u) {
      int sz = g[u].size();
      for (int i = 0; i < sz; ++i) {
        int v = g[u][i];
        // 卷包裹逆时针找圈
        if (!vis[v]) {
          ++face_cnt;
          Polygon poly;
          while (true) {
            vis[v] = 1;
            left[v] = face_cnt;
            int f = e[v].u;
            poly.push_back(p[f]);
            v = prev[v ^ 1];
            if (v == g[u][i]) break;
            assert(vis[v] == 0);
          }
          faces.push_back(poly);
        }
      }
    }
    for (int i = 0; i < face_cnt; ++i) area[i] = GetArea(
        faces[i]);
  }
};
struct Circle { Point o; db r; };
// 切点
std::vector<Point> TagentCP(Circle k1, Point k2) {
  db a = GetLen(k2 - k1.o), b = k1.r * k1.r / a, c = std
      ::sqrt(Max(0., k1.r * k1.r - b * b));
  Point k = GetUnit(k2 - k1.o), m = k1.o + k * b, del =
      Rotate90(k) * c;
  return {m - del, m + del};
}
// 公切线数量
int CheckPosCC(Circle k1, Circle k2) {
  if (Cmp(k1.r, k2.r) == -1) std::swap(k1, k2);
  double dis = k1.o.Dis(k2.o);
  int w1 = Cmp(dis, k1.r + k2.r), w2 = Cmp(dis, k1.r - k2
      .r);
  if (w1 > 0) return 4;
  if (w1 == 0) return 3;
  else if (w2 > 0) return 2;
  else if (w2 == 0) return 1;
  return 0;
}
// 交点
std::vector<Point> GetCC(Circle k1, Circle k2) {
  int pd = CheckPosCC(k1, k2);
  if (pd == 0 || pd == 4) return {};
  double a = (k2.o - k1.o).Abs2();
  double cosA = (k1.r * k1.r + a - k2.r * k2.r) / (2 * k1
      .r * sqrt(std::max(a, 0.0)));
  double b = k1.r * cosA, c = sqrt(std::max(0.0, k1.r *
      k1.r - b * b));
  Point k = (k2.o - k1.o).Unit(), m = k1.o + k * b, del =
       k.Turn90() * c;
  return {m - del, m + del};
}
Circle GetCircle(Point k1, Point k2, Point k3) {
  db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 +
      b1 * b1) * 0.5;
  db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 +
      b2 * b2) * 0.5;
  db d = a1 * b2 - a2 * b1;
  Point o = (Point){k1.x + (c1 * b2 - c2 * b1) / d, k1.y
      + (a1 * c2 - a2 * c1) / d};
  return (Circle){o, GetDis(k1, o)};
}
```

```
db GetMinCircleR(std::vector<Point> p) {
  Point cur = p[0];
  db pro = 10000, ret = inf;
  while (Sgn(pro) > 0) {
    int idx = 0;
    for (int i = 0; i < p.size(); ++i)
      if (GetDis(cur, p[i]) > GetDis(cur, p[idx]))
        idx = i;
    db r = GetDis(cur, p[idx]);
    ret = Min(ret, r);
    cur = cur + (p[idx] - cur) / r * pro;
    pro *= delta;
  }
  return ret;
}
Circle GetMinCircle(std::vector<Point> p) {
  std::random_shuffle(p.begin(), p.end());
  Circle ret = (Circle){p[0], 0.};
  for (int i = 1; i < p.size(); ++i) {
    if (Cmp(GetDis(ret.o, p[i]), ret.r) <= 0) continue;
    ret = (Circle){p[i], 0.};
    for (int j = 0; j < i; ++j) {
      if (Cmp(GetDis(ret.o, p[j]), ret.r) <= 0) continue;
      ret.o = (p[i] + p[j]) * 0.5;
      ret.r = GetDis(ret.o, p[i]);
      for (int k = 0; k < j; ++j) {
        if (Cmp(GetDis(ret.o, p[k]), ret.r) <= 0)
            continue;
        ret = GetCircle(p[i], p[j], p[k]);
      }
    }
  }
  return ret;
}
typedef std::vector<Point> Polygon;
db GetArea(Polygon &poly) {
  db ret = 0.;
  for (int i = 0; i < poly.size(); ++i) ret += poly[i] ^
      poly[(i + 1) % poly.size()];
  return ret * 0.5;
}
Polygon GrahamScan(std::vector<Point> p) {
  Polygon ret;
  if (p.size() < 3) {
    for (Point &v : p) ret.push_back(v);
    return ret;
  }
  int idx = 0;
  for (int i = 0; i < p.size(); ++i)
    if (Cmp(p[i].x, p[idx].x) < 0 || (!Cmp(p[i].x, p[idx
        ].x) && Cmp(p[i].y, p[idx].y) < 0))
      idx = i;
  std::swap(p[0], p[idx]);
  std::sort(p.begin() + 1, p.end(),
    [&](const Point &k1, const Point &k2) {
      db tmp = (k1 - p[0]) ^ (k2 - p[0]);
      if (Sgn(tmp) > 0) return true;
      else if (!Sgn(tmp) && Cmp(GetDis(k1, p[0]), GetDis(
          k2, p[0])) <= 0) return true;
      return false;
    }
  );
  ret.push_back(p[0]);
  for (int i = 1; i < p.size(); ++i) {
    while (ret.size() > 1 && Sgn((ret.back() - ret[ret.
        size() - 2]) ^ (p[i] - ret[ret.size() - 2])) <=
        0) ret.pop_back();
    ret.push_back(p[i]);
  }
  return ret;
}
bool IsIn(Point p, const Polygon &ch) {
  Point base = ch[0];
  if (Sgn((p - base) ^ (ch[1] - p)) > 0 || Sgn((p - base)
      ^ (ch.back() - base)) < 0) return false;
  if (!Sgn((p - base) ^ (ch[1] - p)) && Cmp(GetLen(p -
      base), GetLen(ch[1] - base)) <= 0) return true;
  int idx = std::lower_bound(ch.begin(), ch.end(), p,
    [&](const Point &k1, const Point &k2) {
      return Sgn((k1 - base) ^ (k2 - base)) > 0;
    }
  ) - ch.begin() - 1;
  return Sgn((ch[idx + 1] - ch[idx]) ^ (p - ch[idx])) >=
      0;
}
Polygon Minkowski(const Polygon &k1, const Polygon &k2) {
  int sz1 = k1.size(), sz2 = k2.size();
  std::queue<Point> buf1, buf2;
  for (int i = 0; i < sz1; ++i) buf1.push(k1[(i + 1) %
      sz1] - k1[i]);
  for (int i = 0; i < sz2; ++i) buf2.push(k2[(i + 1) %
      sz2] - k2[i]);
  Polygon ret;
  ret.push_back(k1[0] + k2[0]);
  while (!buf1.empty() && !buf2.empty()) {
    Point tmp1 = buf1.front(), tmp2 = buf2.front();
    if (Sgn(tmp1 ^ tmp2) > 0) {
      ret.push_back(ret.back() + tmp1);
      buf1.pop();
    }
    else {
      ret.push_back(ret.back() + tmp2);
      buf2.pop();
    }
  }
  while (!buf1.empty()) {
    ret.push_back(ret.back() + buf1.front());
    buf1.pop();
```

```
  }
  while (!buf2.empty()) {
    ret.push_back(ret.back() + buf2.front());
    buf2.pop();
  }
  return GrahamScan(ret);
}
db RotateCaliper(Polygon p) {
  db ret = -inf;
  if (p.size() == 3) {
    ret = Max(ret, GetDis(p[0], p[1]));
    ret = Max(ret, GetDis(p[0], p[2]));
    ret = Max(ret, GetDis(p[1], p[2]));
    return ret;
  }
  int cur = 2, sz = p.size();
  for (int i = 0; i < sz; ++i) {
    while (Cmp(std::fabs((p[i] - p[(i + 1) % sz]) ^ (p[
        cur] - p[(i + 1) % sz])), std::fabs((p[i] - p[(i
         + 1) % sz]) ^ (p[(cur + 1) % sz] - p[(i + 1) %
        sz]))) < 0) cur = (cur + 1) % sz;
    ret = Max(ret, GetDis(p[i], p[cur]));
  }
  return ret;
}
```

1.5 动态凸包

```
// CodeForces 70D 动态凸包
#include <bits/stdc++.h>
typedef double db;
const int maxn = 1e5 + 5;
const db eps = 1e-9;
int Sgn(db k) { return fabs(k) < eps ? 0 : (k < 0 ? -1 :
    1); }
int Cmp(db k1, db k2) { return Sgn(k1 - k2); }
struct point { db x, y; };
point operator - (point k1, point k2) { return (point){k1
    .x - k2.x, k1.y - k2.y}; }
point operator + (point k1, point k2) { return (point){k1
    .x + k2.x, k1.y + k2.y}; }
db operator * (point k1, point k2) { return k1.x * k2.x +
     k1.y * k2.y; }
db operator ^ (point k1, point k2) { return k1.x * k2.y -
     k1.y * k2.x; }
db GetLen(point k) { return sqrt(k * k); }
int n;
point basic;
point p[maxn];
std::set<point> set;
bool operator < (point k1, point k2) {
  k1 = k1 - basic; k2 = k2 - basic;
  db ang1 = atan2(k1.y, k1.x), ang2 = atan2(k2.y, k2.x);
  db len1 = GetLen(k1), len2 = GetLen(k2);
  if (Cmp(ang1, ang2) != 0) return Cmp(ang1, ang2) < 0;
  return Cmp(len1, len2) < 0;
}
std::set<point>::iterator Prev(std::set<point>::iterator
    k) {
  if (k == set.begin()) k = set.end();
  return --k;
}
std::set<point>::iterator Next(std::set<point>::iterator
    k) {
  ++k;
  return k == set.end() ? set.begin() : k;
}
bool Query(point k) {
  std::set<point>::iterator it = set.lower_bound(k);
  if (it == set.end()) it = set.begin();
  return Sgn((k - *(Prev(it))) ^ (*(it) - *(Prev(it))))
      <= 0;
}
void Insert(point k) {
  if (Query(k)) return;
  set.insert(k);
  std::set<point>::iterator cur = Next(set.find(k));
  while (set.size() > 3 && Sgn((k - *(Next(cur))) ^ (*(
      cur) - *(Next(cur)))) <= 0) {
    set.erase(cur);
    cur = Next(set.find(k));
  }
  cur = Prev(set.find(k));
  while (set.size() > 3 && Sgn((k - *(cur)) ^ (*(cur) -
      *(Prev(cur)))) >= 0) {
    set.erase(cur);
    cur = Prev(set.find(k));
  }
}
int main() {
  scanf("%d", &n);
  basic.x = basic.y = 0.0;
  for (int i = 1, T; i <= 3; ++i) {
    scanf("%d%lf%lf", &T, &p[i].x, &p[i].y);
    basic.x += p[i].x; basic.y += p[i].y;
  }
  basic.x /= 3.0; basic.y /= 3.0;
  for (int i = 1; i <= 3; ++i) set.insert(p[i]);
  for (int i = 4, T; i <= n; ++i) {
    scanf("%d%lf%lf", &T, &p[i].x, &p[i].y);
    if (T == 1) Insert(p[i]);
    else {
      if (Query(p[i])) printf("YES\n");
      else printf("NO\n");
    }
```

```
  }
  return 0;
}
```

2 数论

2.1 高斯消元

```
const int Mod = 1e9 + 7;
const int maxn = 1e3 + 5;
const double eps = 0.0000001;
const int INF = 0x3f3f3f3f;
int n, m;
double a[maxn][maxn], x[maxn];
bool manySolutionFlag = false, noSolution = false;
void Swap(int i, int j) {
    for (int k = 1; k <= n + 1; k ++)
        swap(a[i][k], a[j][k]);
}
bool Check(int i) {
    bool vis = false;
    for (int j = 1; j <= n; j ++) {
        if(fabs(a[i][j]) >= eps) vis = true;
    }
    if(!vis && fabs(a[i][n + 1]) >= eps) return false;
    return true;
}
void GS() {
    for (int i = 1; i <= n; i ++) {
        bool flag = false;
        for (int j = i; j <= m; j ++) {
            if(a[j][i] != 0) {
                Swap(j, i);
                flag = true;
                break;
            }
        }
        if(!flag) {
            manySolutionFlag = true;
        }
        for (int j = i + 1; j <= m; j ++)
            for (int k = n + 1; k >= i; k --)
                a[j][k] = a[j][k] * 1. - a[i][k] * (a[j][
                    i] * 1./a[i][i] * 1.) * 1.;

    }
    for (int i = 1; i <= m; i ++) {
        if(!Check(i)) {
            noSolution = true;
            return ;
        }
    }
    for (int i = n; i >= 1; i --) {
        for (int j = i + 1; j <= n; j ++) {
            a[i][n + 1] = a[i][n + 1] - a[i][j] * x[j];
            a[i][j] = 0;
```

```
        }
        x[i] = a[i][n + 1] * 1./a[i][i] * 1.;
    }
}
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i ++)
        for (int j = 1; j <= n + 1; j ++)
            cin >> a[i][j];
    GS();
    if(noSolution) cout << "No solutions" << endl;
    else if(manySolutionFlag) cout << "Many solutions" <<
          endl;
    else {
        for (int i = 1; i <= n; i ++)
            cout << (int)(x[i] +0.5) << endl;
    }
    return 0;
}
```

2.2 逆元

```
int inv[maxn];// 递推打表
void getInv(int n, int m) {
    inv[1] = 1;
    for (int i = 2; i <= n; i ++)
        inv[i] = (long long) (m - m/i) * inv[m%i] % m;
}
long long ex_gcd(long long a, long long b, long long &x,
     long long &y) { //扩展欧几里德求逆元
    if(!b) {
        x = 1; y = 0;
        return a;
    }
    long long d = ex_gcd(b, a%b, x, y);
    long long t = x;
    x = y; y = t - (a/b) * y;
    return d;
}
long long getInv(long long a, long long p) {
    long long x, y;
    ex_gcd(a, p, x, y);
    return (x % p + p) % p;
}
long long Ksm(long long a, long long b, long long mod) {
     // 递推打阶乘逆元表
    long long res = 1;
    while(b) {
        if(b & 1) res = res * a % mod;
        a = a % a % mod;
        b >>= 1;
    }
    return res;
}
long long Fac[maxn], inv[maxn];
void init() {
    Fac[0] = 1;
    for (int i = 1; i <= maxn; i ++)
        Fac[i] = (Fac[i-1] * i) % mod;
    inv[maxn] = Ksm(Fac[maxn], mod-2);
    for (int i = maxn - 1; i >= 0; i --)
        inv[i] = inv[i+1] * (i+1) % mod;
}
```

2.3 线性基

```
struct LB{
    long long b[35], nb[35], tot;
    bool flag;
    LB () {  // 初始化
        memset(b, 0, sizeof(b));
        flag = false;
    }
    LB(const LB& a) {
        for (int i = 0; i < 35; i ++)
            b[i] = a.b[i];
        flag = a.flag;
    }
    void Ins(long long x) { //插入
        for (int i = 34; i >= 0; i --)
            if(x & (1ll << i)) {
                if(!b[i]) { b[i] = x; return; }
                x ^= b[i];
            }
        flag = true; //能xor出0
    }
    bool Fin(long long x) {
        if(x == 0 && flag) return true;
        for (int i = 34; i >= 0; i --)
            if(x >> i) x ^= b[i];
        return x == 0;
    }
    long long getMax(long long x) { // 得到最大值
        long long res = x;
        for (int i = 34; i >= 0; i --)
            res = max(res, res ^ b[i]);
        return res;
    }
    long long getMin(long long x) { // 得到最大值
        long long res = x;
        for (int i = 0; i <= 34; i ++)
```

```
            if(b[i]) res ^= b[i];
        return res;
    }
    long long ReBuild() { // 重新Build 为下面的Kth
        for (int i = 34; i >= 0; i --) {
            if(b[i] == 0) continue;
            for (int j = i - 1; j >= 0; j --) {
                if(b[j] == 0) continue;
                if(b[i] & (1ll << j)) b[i] ^= b[j];
            }
        }
        for (int i = 0; i <= 34; i ++)
            if(b[i]) nb[tot++] = b[i];
    }
    long long Kth_Max(long long k) { // 得到第k小的数, k
        >1,
        if(flag) k --;
        if(k == 0) return 0;
        long long res = 0;
        if(k >= (1ll << tot)) return -1;
        for (int i = 34; i >= 0; i --)
            if(k & (1ll << i)) res ^= nb[i];
        return res;
    }
    LB Corss(LB k) { //求交集
        LB res, tmp = k;
        for (int i = 0; i < 35; i ++) {
            long long x = b[i], y = 0;
            bool vis = false;
            for (int j = 34; j >= 0; j --) {
                if(x >> j) {
                    if(k.b[j]) x ^= k.b[j], y ^= tmp.b[j
                        ];
                    else {
                        k.b[j] = x;
                        tmp.b[j] = y;
                        vis = true;
                        break;
                    }
                }
            }
            if(!vis) res.b[i] = y;
        }
        return res;
    }
    LB Merge(LB u) { //合并两个线性集
        LB w = *this;
        for (int i = 34; i >= 0; i--) {
            if(u.b[i] == 0) continue;
            w.Ins(u.b[i]);
        }
        return w;
    }
};
```

2.4 约瑟夫环

```
long long Josephus(long long N, long long K) {//N 个人,
    第K 淘汰
    if(N == 1) return 0;
    if(N < K) {
        long long ret = 0;
        for (long long i = 2; i <= N; i ++)
            ret = (ret + K) % i;
        return ret;
    }
    long long ret = Josephus(N - N/K, K);
    if(ret < N % K) ret = ret - N % K + N;
    else ret = ret - N % K + (ret - N % K) / (K - 1);
    return ret;
}
```

2.5 欧拉函数

```
int getPhi(int n) {
    int rea = n;
    for (int i = 2; i * i <= n; i ++) {
        if (n % i == 0) {
            rea = rea - rea / i;
            while(n % i == 0) n /= i;
        }
    }
    if(n > 1) rea = rea - rea / n;
    return rea;
}
int phi[maxn];//递推
void getPhi() {
    for (int i = 1; i < maxn; i ++)
        phi[i] = i;
    for (int i = 2; i < maxn; i ++)
        if(phi[i] == i)
            for (int j = i; j < maxn; j += i)
                phi[j] = (phi[j] / i) * (i-1);
}
```

2.6 杜教筛

```
const int M = 5e6;
const int Mod = 1e9 +7;
```

```
const double eps = 0.000000001;
long long phi[M + 30], prim[M + 30], tot;
bool mark[M + 30];
map<long long, long long> m;
#define inv_2 (Mod+1)/2
long long Add(long long a, long long b) {
    long long c = (a + b) % Mod;
    if(c >= Mod) return c - Mod;
    if(c < 0) return c + Mod;
    return c;
}
void init() {
    phi[1] = 1;
    for(long long i = 2; i <= M; i ++) {
        if(!mark[i]) {
            prim[++tot] = i;
            phi[i] = i - 1;
        }
        for (long long j = 1; j <= tot; j ++) {
            if(i * prim[j] > M) break;
            mark[i * prim[j]] = 1;
            if(i % prim[j] == 0) {
                phi[i * prim[j]] = phi[i] * prim[j];
                break;
            }
            phi[i *prim[j]] = phi[i] * phi[prim[j]];
        }
    }
    for (int i = 1; i <= M; i ++) phi[i] = Add(phi[i-1],
         phi[i]);
}
long long getPhi(long long n) {
    if(n <= M) return phi[n];
    if(m[n]) return m[n];
    long long ans;
    ans = 1LL * n % Mod * (n % Mod + 1) % Mod * inv_2 %
         Mod;
    for (long long l = 2, r; l <= n; l = r + 1) {
        r = n/(n/l);
        ans = (ans - (r - l + 1) % Mod * getPhi(n/l) %
             Mod + Mod) % Mod;
    }
    return m[n] = ans;
}
long long solve(long long n) {
    long long ans = 0;
    for (long long l = 1, r; l <= n; l = r + 1) {
        r = n/(n/l);
        ans = Add(ans, 1ULL * (n/l) % Mod * (n/l) % Mod *
              (Add(getPhi(r) % Mod, -getPhi(l-1) % Mod)
             ) % Mod);
    }
    return ans;
}
```

```
int main()
{
    init();
    long long n;
    cin >> n;
    m.clear();
    cout << solve(n) << endl;
    return 0;
}
/*
gcd之和
gcd(i,j)(1<=i<=n)(1<=j<=m)
* /
```

2.7 数论函数打表

```
int phi[maxn], prime[maxn], tot;//线性打欧拉函数
bool vis[maxn];
void init() {
    phi[1] = 1;
    for (int i = 2; i < maxn; i ++) {
        if(!vis[i]) {
            prime[++tot] = i;
            phi[i] = i - 1;
        }
        for (int j = 1; j <= tot; j ++) {
            if(i * prime[j] >= maxn) break;
            vis[i*prime[j]] = 1;
            if(i % prime[j] == 0) {
                phi[i*prime[j]] = phi[i] * prime[j];
                break;
            }
            phi[i*prime[j]] = phi[i] * phi[prime[j]];
        }
    }
}
bool vis[maxn];
int mu[maxn], prime[maxn];
void Mobius() {//线性打莫比乌斯函数
    mu[1] = 1;
    int tot = 0;
    for (int i = 2; i < maxn; i ++) {
        if(!vis[i]) {
            prime[tot++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < tot; j ++) {
            if(i * prime[j] >= maxn) break;
            vis[i*prime[j]] = true;
            if(i % prime[j] == 0) {
                mu[i*prime[j]] = 0;
```

```
                break;
            }
            mu[i*prime[j]] = -mu[j];
        }
    }
}
```

2.8 扩展卢卡斯

```
long long Pre[maxn];
long long extend_gcd(long long a, long long b, long long
     &x, long long &y) {
    if(!b) {
        x = 1; y = 0;
        return a;
    }
    long long d = extend_gcd(b, a % b, x, y);
    long long t = x;
    x = y; y = t - (a / b) * y;
    return d;
}
long long mul(long long a, long long b, long long P){
    long long L = a * (b >> 25LL) % P * (1LL << 25) % P;
    long long R = a * (b & ((1LL << 25) - 1)) % P;
    return (L + R) % P;
}
long long Pow(long long a, long long b, long long P) {
    long long ans = 1; a %= P;
    while(b) {
        if(b & 1) ans = mul(ans, a, P);
        a = mul(a, a, P);
        b >>= 1;
    }
    return ans;
}
long long getInv(long long a, long long p) {
    long long x, y;
    extend_gcd(a, p, x, y);
    x = (x % p + p) % p;
    return x;
}
long long CRT(long long m, long long p, long long P) {
    return mul(mul(m, (P / p), P), getInv(P / p, p), P);
}
void init(long long pi, long long pk) {
    Pre[0] = 1;
    for (int i = 1; i <= pk; i ++) {
        Pre[i] = Pre[i - 1];
        if(i % pi) Pre[i] = mul(Pre[i], i, pk);
    }
}
long long Mul(long long n, long long pi, long long pk) {
    if(n <= 1) return 1;
    long long ans = Pow(Pre[pk], n / pk, pk);
    if(n % pk) ans = mul(ans, Pre[n % pk], pk);
    return mul(ans, Mul(n / pi, pi, pk), pk);
}
long long C(long long n, long long m, long long pi, long
     long pk) {
    if(n < m) return 0;
    init(pi, pk);
    long long r = 0;
    for(long long i = n; i; i /= pi) r += i / pi;
    for(long long i = m; i; i /= pi) r -= i / pi;
    for(long long i = n - m; i; i /= pi) r -= i / pi;
    long long a = Mul(n, pi, pk);
    long long b = getInv(Mul(m, pi, pk), pk);
    long long c = getInv(Mul(n - m, pi, pk), pk);
    long long ans = mul(mul(a, b, pk), c, pk);
    return mul(ans, Pow(pi, r, pk), pk);
}
long long ex_lucas(long long n, long long m, long long P)
      {
//C_n^m %p
    long long ans = 0;
    long long p = P;
    for (int i = 2; i <= P; i ++) {
        if(p % i == 0) {
            long long pi = i, pk = 1;
            while(p % i == 0) {
                p /= i;
                pk *= i;
            }
            ans = (ans + CRT(C(n, m, pi, pk), pk, P)) % P
                 ;
        }
    }
    return ans;
}
```

2.9 扩展中国剩余定理

```
long long M[maxn], C[maxn];//模数,  余数
long long mul(long long a, long long b, long long p) {
    if(b < 0) b = -b;
    long long ans = 0;
    while(b) {
        if(b & 1) ans = (ans + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return ans;
```

```
}
long long gcd(long long a, long long b) {
    return !b ? a : gcd(b, a %b);
}
long long exgcd(long long a, long long b, long long &x,
     long long &y) {
    if(!b) {
        x = 1;
        y = 0;
        return a;
    }
    long long d = exgcd(b, a % b, x, y);
    long long t = x;
    x = y;
    y = t - (a / b) * y;
    return d;
}
long long getInv(long long a, long long p) {
    long long x, y;
    exgcd(a, p, x, y);
    x = (x % p + p) % p;
    return x;
}
long long exCrt() {
    for (long long i = 2; i <= n; i ++) {
        long long M1 = M[i - 1], M2 = M[i];
        long long C1 = C[i - 1], C2 = C[i];
        long long T = gcd(M1, M2);
        long long t = (C2 - C1 % M2 + M2) % M2;
        if(t % T) return -1;
        M[i] = M1 / T * M2;
        C[i] = mul(getInv(M1 / T, M2 / T), t / T, (M2 / T
             ));
        C[i] = C[i] * M1 + C1;
        C[i] = (C[i] % M[i] + M[i]) % M[i];
    }
    return C[n];
}
```

2.10 快速幂

```
long long Ksm(long long a, long long b, long long mod) {
    long long res = 1;
    while(b) {
        if(b & 1) res = Ksc(res, a, mod);
        a = Ksc(a, a, mod);
        b >>= 1;
    }
    return res;
}
```

2.11 快速傅里叶变换

```
const int maxn = 5e5 + 5;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 + 7;
typedef complex<double> cp;
const double PI = acos(-1);
char sa[maxn], sb[maxn];
int n = 1, lena, lenb, res[maxn];
cp a[maxn], b[maxn], omg[maxn], inv[maxn];
void init() {
    for (int i = 0; i < n; i ++) {
        omg[i] = cp(cos(2*PI*i/n), sin(2*PI*i/n));
        inv[i] = conj(omg[i]);
    }
}
void fft(cp *a, cp *omg) {
    int lim = 0;
    while((1<<lim) < n) lim++;
    for (int i = 0; i < n; i ++) {
        int t = 0;
        for (int j = 0; j < lim; j ++)
            if((i>>j) & 1) t |= (1<<(lim-j-1));
        if(i < t) swap(a[i], a[t]);
    }
    for (int l = 2; l <= n; l *= 2) {
        int m = l / 2;
        for (cp *p = a; p != a + n; p += l)
            for (int i = 0; i < m; i ++) {
                cp t = omg[n/l*i] * p[i+m];
                p[i+m] = p[i] - t;
                p[i] += t;
            }
    }
}
int main() {
    scanf("%d", &n);
    scanf("%s%s", sa, sb);
    lena = lenb = n;
    n = 1;
    while(n < lena + lenb) n <<= 1;
    for (int i = 0; i < lena; i ++)
        a[i].real(sa[lena-1-i] - '0');
    for (int i = 0; i < lenb; i ++)
        b[i].real(sb[lenb-1-i] - '0');
    init();
    fft(a, omg);fft(b, omg);
    for (int i = 0; i < n; i ++)
        a[i] *= b[i];
    fft(a, inv);
    for (int i = 0; i < n; i ++) {
        res[i] += floor(a[i].real()/n + 0.5);
        res[i+1] += res[i] / 10;
```

```
        res[i] %= 10;
    }
    int pos = n - 1;
    while(!res[pos]) pos--;
    for (int i = pos; i >= 0; i --) printf("%d", res[i]);
    puts("");
    return 0;
}
```

2.12 快速乘

```
long long Ksc(long long a, long long b, long long mod) {
     //普通版
    long long ans = 0;
    while(b) {
        if(b & 1) ans = (ans + a ) % mod;
        b >>= 1;
        a = (a + a) % mod;
    }
    return ans;
}
long long Ksc(long long a, long long b, long long mod){//
     快速版
    long long L = a * (b >> 25ll) % mod * (1ll << 25) %
         mod;
    long long R = a * (b & ((1ll << 25) - 1)) % mod;
    return (L + R) % mod;
}
long long Ksc(long long a, long long b, long long mod) {
     //精确版
    a %= mod, b %= mod;
    return ((a * b - (long long)((long long)((long double
         )a / mod * b + 1e-3) * mod)) % mod + mod) % mod;
}
```

2.13 卢卡斯定理

```
long long  exgcd(long long a, long long b, long long &x,
     long long &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    long long ans = exgcd(b, a % b, x, y);
    long long temp = x;
    x = y;
    y = temp - (a / b) * y;
    return ans;
}
long long inv(long long a) {
    long long x, y;
    long long t = exgcd(a, M, x, y);
    if (t != 1) {
        return -1;
    }
    return (x % M + M) % M;
}
long long fac[maxn];
void getfac() {
    fac[0] = 1;
    for (int i = 1; i < maxn; i++) {
        fac[i] = fac[i - 1] * i % M;
    }
}
long long C(long long n, long long m) {
    if (n < 0 || m < 0|| n < m) {
        return 0;
    }
    return fac[n] * inv(fac[m]) % M * inv(fac[n - m]) % M
         ;
}
long long lucas(long long n, long long m) {
    if (m == 0) {
        return 1;
    }
    return (lucas(n / M, m / M) * C(n % M, m % M)) % M;
}
```

2.14 十进制快速幂

```
const int maxn = 1e6 + 5; //矩阵
long long mod;
struct Matrix{
    long long mat[2][2];
    Matrix() {memset(mat, 0, sizeof(mat));};
    void init() {
        mat[0][0] = mat[1][1] = 1;
    }
    void init(long long a, long long b) {
        mat[0][0] = 0; mat[0][1] = b;
        mat[1][0] = 1; mat[1][1] = a;
    }
    void operator = (Matrix x) {
        for (int i = 0; i <= 1; i ++)
            for (int j = 0; j <= 1; j ++)
                mat[i][j] = x.mat[i][j];
    }
};
void Print(Matrix x) {
```

```
    for (int i = 0; i <= 1; i ++) {
        for (int j = 0; j <= 1; j ++)
            cout << x.mat[i][j] << " ";
        cout << endl;
    }
}
Matrix operator * (Matrix x, Matrix y) {
    Matrix t;
    for (int i = 0; i <= 1; i ++)
        for (int j = 0; j <= 1; j ++)
            for (int k = 0; k <= 1; k ++)
                t.mat[i][j] = (t.mat[i][j] + x.mat[i][k]
                    * y.mat[k][j]) % mod;
    return t;
}
Matrix Ksm(Matrix x, long long b) {
    Matrix t; t.init();
    while(b) {
        if(b & 1) t = t * x;
        x = x * x;
        b >>= 1;
    }
    return t;
}
int main() {
    long long x0, x1, a, b;
    scanf("%lld %lld %lld %lld", &x0, &x1, &a, &b);
    char s[maxn];
    scanf("%s%lld", s, &mod);
    int len = strlen(s);
    reverse(s, s+len);
    Matrix t, ans; t.init(a, b);
    ans.mat[0][0] = x0; ans.mat[0][1] = x1;
    Matrix res;
    res.init();
    for (int i = 0; i < len; i ++) {
        res = res * Ksm(t, s[i]-'0');
        t = Ksm(t, 10);
    }
    ans = ans * res;
    printf("%lld\n", ans.mat[0][0]);
    return 0;
}
```

2.15 二次剩余

```
struct T{
    long long p, d;
};
long long Ksm(long long a, long long b, long long p) {
    long long res = 1;
    while(b) {
        if(b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}
long long w;
T Mul_er(T a, T b, long long p) {//二次域乘法
    T ans;
    ans.p = (a.p * b.p + a.d * b.d % p * w % p) % p;
    ans.d = (a.p * b.d % p + a.d * b.p % p) % p;
    return ans;
}
T Ksm_er(T a, long long b, long long p) {//二次域快速幂
    T ans;
    ans.p = 1; ans.d = 0;
    while(b) {
        if(b & 1) ans = Mul_er(ans, a, p);
        a = Mul_er(a, a, p);
        b >>= 1;
    }
    return ans;
}
long long Legendre(long long a, long long p) {//求勒让德
    符号
    return Ksm(a, (p-1)>>1, p);
}
long long Recever(long long a, long long p) {
    a %= p;
    if(a < 0) a += p;
    return a;
}
long long solve(long long n, long long p) {
    if(n % p == 0) return 0;
    if(p == 2) return 1;
    if(Legendre(n, p) + 1 == p) return -1;
    long long a = -1, t;
    while(1) {
        a = rand() % p;
        t = a * a - n;
        w = Recever(t, p);
        if(Legendre(w, p) + 1 == p) break;
    }
    T tmp;
    tmp.p = a; tmp.d = 1;
    T ans = Ksm_er(tmp, (p+1)>>1, p);
    return ans.p;
}
```

2.16 三分

```
double Com(double X) {
    return sqrt((X - x) * (X - x) + (a * X * X + b * X +
         c - y) * (a * X * X + b * X + c - y));
}
void Binary(double l, double r) {
    if(l + eps <= r) {
        double lm, rm;
        double k = r - l;
        lm = l + (1./3) * k;
        rm = r - (1./3) * k;
        if(fabs(Com(lm) - Com(rm)) <= eps) {
             printf("%.31f\n", Com(lm));
             return ;
        }
        if(Com(lm) < Com(rm))
            Binary(l, rm);
        else Binary(lm, r);
    }
}
```

2.17 min25 筛

```
#define inv_2 (Mod+1)/2
#define inv_6 (Mod+1)/6
long long sqr, m, w[maxn], g[maxn], h[maxn];
long long sumg[maxn], sumh[maxn], id1[maxn], id2[maxn];
long long prim[maxn], tot;
bool mark[maxn];
long long Add(long long a, long long b) {
    return (a + b) % Mod;
}
long long Sup(long long a, long long b) {
    return (a - b + Mod) % Mod;
}
long long Pow(long long a, long long b) {
    long long res = 1;
    while(b) {
        if(b & 1) res = res * a % Mod;
        a = a * a % Mod;
        b >>= 1;
    }
    return res;
}
void init(long long n) {
    mark[1] = 1;
    for (long long i = 2; i <= n; i ++) {
        if(!mark[i]) {
            prim[++tot] = i;
            sumg[tot] = (sumg[tot-1] + i * i) % Mod;
            sumh[tot] = (sumh[tot-1] + i) % Mod;
        }
        for (long long j = 1; j <= tot; j ++) {
            if(i * prim[j] > n) break;
            mark[i * prim[j]] = 1;
            if(i % prim[j] == 0) break;
        }
    }
}
void GetW(long long n) {
    for (long long i = 1, j; i <= n; i = j + 1) {
        j = n / (n / i);
        w[++m] = n / i;
        long long t = w[m] % Mod;
        g[m] = t * (t + 1) % Mod * ((2LL * t + 1) % Mod)
             % Mod * inv_6 % Mod;
        g[m] --;
        h[m] = t * (t + 1) % Mod * inv_2 % Mod;
        h[m] --;
        if(w[m] <= sqr) id1[w[m]] = m;
        else id2[n/w[m]] = m;
    }
}
void GetG(long long n) {
    for (long long i = 1; i <= tot; i ++) {
        for (long long j = 1; j <= m && prim[i] * prim[i]
              <= w[j]; j ++) {
            long long d = w[j] / prim[i];
            long long id = d <= sqr ? id1[d] : id2[n/d];
            g[j] = Sup(g[j], prim[i] * prim[i] % Mod * ((
                 g[id] - sumg[i-1] + Mod) % Mod) % Mod);
            h[j] = Sup(h[j], prim[i] * ((h[id] - sumh[i
                 -1] + Mod) % Mod) % Mod);
        }
    }
}
long long S(long long x, long long y, long long n) {
    if(x <= prim[y-1] || x <= 1) return 0;
    long long id = x <= sqr ? id1[x] : id2[n/x];
    long long res = (g[id] - h[id] + Mod - sumg[y-1] +
         sumh[y-1] + Mod) % Mod;
    for (long long i = y; i <= tot && prim[i] * prim[i]
          <= x; i ++) {
        long long t = prim[i];
        for (long long j = 1; t <= x; j ++, t = t * prim[
             i]) {
            long long p1 = t % Mod;
            res = Add(res, p1 * (p1 - 1) % Mod * (S(x/t,
                 i+1, n) + (j != 1)) % Mod);
        }
    }
    return res % Mod;
}
int main(int argc, char *args[]) {
    long long n;
    scanf("%lld", &n);
```

```
    sqr = sqrt(n);
    init(sqr);
    GetW(n);
    GetG(n);
    printf("%lld\n", (S(n, 1, n) + 1) % Mod);
    return 0;
}
```

2.18 SG 函数

```
int SG[maxn], S[maxn];
int f[maxn];
void ff() {//f是每一次走的步数
    f[0] = 1;
    for (int i = 1; i <= 10; i ++)
        f[i] = f[i-1] * 2;
}
void getSG(int n) {
    ff();
    for (int i = 1; i <= n; i ++) {
        memset(S, 0, sizeof(S));
        for (int j = 0; f[j] <= i && j <= 10; j ++)
            S[SG[i-f[j]]] = 1;
        for (int j = 0; ; j ++)
            if(!S[j]) {
                SG[i] = j;
                break;
            }
    }
}
int main() {
    int n;
    getSG(1000);
    while(cin >> n) {
        if(SG[n]) cout << "Kiki\n";
        else cout << "Cici\n";
    }
    return 0;
}
```

2.19 Pollard Rho

```
long long factor[1000005];
int tot;
const int S=50;
long long Ksc(long long a, long long b, long long mod) {
    a %= mod, b %= mod;
    return ((a * b - (long long)((long long)((long double
        )a / mod * b + 1e-3) * mod)) % mod + mod) % mod;
}
long long Ksm(long long a, long long b, long long mod) {
    long long res = 1;
    while(b) {
        if(b & 1) res = Ksc(res, a, mod);
        a = Ksc(a, a, mod);
        b >>= 1;
    }
    return res;
}
bool check(long long a, long long n, long long x, long
     long t) {
    long long ret = Ksm(a, x, n);
    long long last = ret;
    for(int i = 1; i <= t; i ++) {
        ret = Ksc(ret, ret, n);
        if(ret == 1 && last != 1 && last != n-1) return
            true;//是合数
        last = ret;
    }
    if(ret != 1) return true;
    return false;
}
bool Miller_Rabin(long long n) {//判素数
    if(n < 2) return false;
    if(n == 2) return true;
    if((n&1) == 0)return false;
    long long x = n - 1;
    long long t = 0;
    while((x&1) == 0) {
        x >>= 1;
        t ++;
    }
    for(int i = 0; i < S; i ++) {
        long long a = rand() % (n-1) + 1;
        if(check(a, n, x, t))//如果检查出来是合数
        return false;
    }
    return true;
}
long long gcd(long long a,long long b) {
    if(a == 0) return 1;
    if(a < 0) return gcd(-a, b);
    while(b) {
        long long t = a % b;
        a = b;
        b = t;
    }
    return a;
}
long long pollard_rho(long long x,long long c) {
    long long i = 1, k = 2;
    long long x0 = rand() % x;
    long long y = x0;
```

```
    while(1) {
        i ++;
        x0 = (Ksc(x0, x0, x) + c) % x;
        long long d = gcd(y-x0, x);
        if(d != 1 && d != x) return d;
        if(y == x0) return x;
        if(i == k) {
            y = x0;
            k += k;
        }
    }
}
void findphi(long long n) {
    if(Miller_Rabin(n)) {
        factor[tot++] = n;
        return;
    }
    long long p = n;
    while(p >= n) {
        p=pollard_rho(p, rand()%(n-1)+1);
    }
    findphi(p);
    findphi(n/p);
}
int main() {
    long long n;
    while(scanf("%I64d",&n)!=EOF) {
        tot=0;
        findphi(n);
        for(int i=0;i<tot;i++)
        printf("%I64d",factor[i]),printf("\n");
        if(Miller_Rabin(n))printf("yes\n");
        else printf("no\n");
    }
    return 0;
}
```

2.20 NTT

```
const int maxn = 1e5 + 5;
const int inf = 0x3f3f3f3f;
const int mod = 998244353;
#define Mod(x) ((x)>=mod?(x)-mod:(x))
#define g 3
int rnk[maxn];
long long a[maxn], b[maxn];
long long Ksm(long long a, long long b) {
    long long res = 1;
    while(b) {
        if(b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
void init(int n) {
    memset(rnk, 0, sizeof(rnk));
    int lim = 0;
    while((1<<lim) < n) lim ++;
    for (int i = 0; i < n; i ++)
        rnk[i] = (rnk[i>>1]>>1) | ((i&1) << (lim-1));
}
void ntt(long long *a, int op, int n) {
    for (int i = 0; i < n; i ++)
        if(i < rnk[i]) swap(a[i], a[rnk[i]]);
    for (int i = 2; i <= n; i <<= 1) {
        int nw = Ksm(g, (mod-1)/i);
        if(op == -1) nw = Ksm(nw, mod-2);
        for (int j = 0, m = i >> 1; j < n; j += i)
            for (int k = 0, w = 1; k < m; k ++) {
                int t = 1ll * a[j+k+m] * w % mod;
                a[j+k+m] = Mod(a[j+k]-t+mod);
                a[j+k] = Mod(a[j+k]+t);
                w = 1ll * w * nw % mod;
            }
    }
    if(op == -1)
        for (int i = 0, inv = Ksm(n, mod-2); i < n; i ++)
            a[i] = 1ll * a[i] * inv % mod;
}
char s1[maxn], s2[maxn];
long long ans[maxn];
int main() {
    scanf("%s", s1);
    scanf("%s", s2);
    int len1 = strlen(s1), len2 =strlen(s2);
    int n = 1;
    while(n < len1 + len2) n <<= 1;
    init(n);
    for (int i = 0; i < len1; i ++) a[len1-i-1] = s1[i]-'
        0';
    for (int i = 0; i < len2; i ++) b[len2-i-1] = s2[i]-'
        0';
    ntt(a, 1, n); ntt(b, 1, n);
    for (int i = 0; i < n; i ++)
        a[i] = (1ll * a[i] * b[i]) % mod;
    ntt(a, -1, n);
    for (int i = 0; i < n; i ++)
        cout << a[i] << " ";
    cout << endl;
    for (int i = 0; i < n; i ++) {
        ans[i+1] += ans[i] / 10;
        ans[i] %= 10;
    }
    int pos = n-1;
```

```
    while(!a[pos]) pos--;
    for (int i = pos; i >= 0; i --) cout << a[i];
    cout << endl;
    return 0;
}
//三模NTT
#define long long long long
const long long maxn = 3 * 1e6 + 10;
#define swap(x,y) x ^= y, y ^= x, x ^= y
using namespace std;
long long a[maxn], b[maxn];
long long Mul(long long a, long long b, long long mod) {
    a %= mod, b %= mod;
    return ((a * b - (long long)((long long)((long double
        )a / mod * b + 1e-3) * mod)) % mod + mod) % mod;
}
long long Ksm(long long a, long long p, long long mod) {
    long long base = 1;
    while(p) {
        if(p & 1) base = 1ll * a * base % mod;
        a = 1ll * a * a % mod; p >>= 1;
    }
    return base % mod;
}
namespace NTT{

    const long long P1 = 469762049, P2 = 998244353, P3 =
        1004535809, g = 3;
    const long long PP = 1ll * P1 * P2;
    long long n, m, p, len = 1, lim;
    long long tmp1[maxn], tmp2[maxn], ans[3][maxn], r[
        maxn];
    long long res[maxn], tmp[maxn], base[maxn];
    /*
        传的参数n,m都比实际个数少一
        n--;m--;
        输入两个数n=1
        输入一个数n=0;
    */
    void init(long long n) { //初始化, 传入alen+blen,得到
        最小的len
        len = 1; lim = 0;
        while(len <= n) len <<= 1, lim++;
        for(long long i = 0; i <= len; i++) r[i] = (r[i
            >> 1] >> 1) | ((i & 1) << (lim - 1));
    }
    void ntt_Mod(long long *a, const long long n, const
        long long type, const long long mod) { //ntt
        for(long long i = 0; i < n; i++) if(i < r[i])
            swap(a[i], a[r[i]]);
        for(long long mid = 1; mid < n; mid <<= 1) {
            long long W = Ksm(type == 1 ? g : Ksm(g, mod
                - 2, mod) , (mod - 1) / (mid << 1), mod)
                ;
            for(long long j = 0; j < n; j += (mid << 1))
                {
                long long w = 1;
                for(long long k = 0; k <mid; k++, w = 1ll
                    * w * W % mod) {
                    long long x = a[j + k], y = 1ll * w *
                        a[j + k + mid] % mod;
                    a[j + k] = (x + y) % mod,
                    a[j + k + mid] = (x - y + mod) % mod;
                }
            }
        }
        if(type == -1) {
            long long inv = Ksm(n, mod - 2, mod);
            for(long long i = 0; i < n; i++)
                a[i] = 1ll * a[i] * inv % mod;
        }
    }
    void Out(long long *a, long long len) {
        for (int i = 0; i <= len; i ++)
            cout << a[i] << " ";
        cout << endl;
    }
    int ntt_Mul(long long *a, long long *b, long long
        alen, long long blen, long long mod) {
        init(alen + blen);
        memcpy(tmp1, a, sizeof(tmp1)); memcpy(tmp2, b,
            sizeof(tmp2));
        ntt_Mod(tmp1, len, 1, P1); ntt_Mod(tmp2, len, 1,
            P1);
        for(long long i = 0; i <= len; i++) ans[0][i] = 1
            ll * tmp1[i] * tmp2[i] % P1;
        memcpy(tmp1, a, sizeof(tmp1)); memcpy(tmp2, b,
            sizeof(tmp2));
        ntt_Mod(tmp1, len, 1, P2); ntt_Mod(tmp2, len, 1,
            P2);
        for(long long i = 0; i <= len; i++) ans[1][i] = 1
            ll * tmp1[i] * tmp2[i] % P2;
        memcpy(tmp1, a, sizeof(tmp1)); memcpy(tmp2, b,
            sizeof(tmp2));
        ntt_Mod(tmp1, len, 1, P3); ntt_Mod(tmp2, len, 1,
            P3);
        for(long long i = 0; i <= len; i++) ans[2][i] = 1
            ll * tmp1[i] * tmp2[i] % P3;
        ntt_Mod(ans[0], len, -1, P1);
        ntt_Mod(ans[1], len, -1, P2);
        ntt_Mod(ans[2], len, -1, P3);
        for(long long i = 0; i <= alen + blen; i++) {
            long long t = (Mul(1ll * ans[0][i] * P2 % PP,
                Ksm(P2 % P1, P1 - 2, P1), PP) +
                  Mul(1ll * ans[1][i] * P1 % PP, Ksm(P1
                      % P2, P2 - 2, P2), PP) ) % PP;
            long long K = ((ans[2][i] - t) % P3 + P3) %
                P3 * Ksm(PP % P3, P3 - 2, P3) % P3;
```

```
            a[i] = (t % mod + ((K % mod) * (PP % mod)) %
                 mod ) % mod;
        }
        return alen + blen;
    }
    int ntt_Ksm(long long *a, long long b, int blen, long
          long mod) {
        memcpy(base, a, sizeof(base));
        memset(a, 0, maxn*sizeof(a));
        a[0] = 1; int alen = 0;
        while(b) {
            if(b & 1) alen = ntt_Mul(a, base, alen, blen,
                  mod);
            memcpy(tmp, base, sizeof(tmp));
            blen = ntt_Mul(base, tmp, blen, blen, mod);
            b >>= 1;
        }
        return alen;
    }
}
int main() {
    long long n, m, p;
    scanf("%lld %lld %lld", &n, &m, &p);
    for(long long i = 0; i <= n; i++) scanf("%lld", &a[i
         ]);
    for(long long i = 0; i <= m; i++) scanf("%lld", &b[i
         ]);
    NTT::ntt_Mul(a, b, n, m, p);
    for (int i = 0; i <= n + m; i ++)
        printf("%lld ", a[i]);
    printf("\n");
    return 0;
}
```

2.21 Miller Rabin

```
#define random(a, b) (((double)rand()/RAND_MAX)*(b-a)+a)
long long Ksc(long long a, long long b, long long mod) {
    a %= mod, b %= mod;
    return ((a * b - (long long)((long long)((long double
         )a / mod * b + 1e-3) * mod)) % mod + mod) % mod;
}
long long Ksm(long long a, long long b, long long mod) {
    long long res = 1;
    while(b) {
        if(b & 1) res = Ksc(res, a, mod);
        a = Ksc(a, a, mod);
        b >>= 1;
    }
    return res;
}
```

```
bool Miller_Rabin(long long n) {
    if(n <= 2) {
        if(n == 2) return true;
        return false;
    }
    if(n % 2 == 0) return false;
    long long u = n - 1;
    while(u % 2 == 0) u /= 2;
    int S = 100;
    srand((long long)time(0));
    for (int i = 1; i <= S; i ++){
        long long a = rand() % (n - 2)+ 2;
        long long x = Ksm(a, u, n);
        while(u < n) {
            long long y = Ksm(x, 2, n);
            if(y == 1 && x != 1 && x != n - 1)
                return false;
            x = y;
            u = u * 2;
        }
        if(x != 1) return false;
    }
    return true;
}
```

2.22 BigInteger

```
#define MAXN 999
#define MAXSIZE 100240
#define DLEN 3
struct BigInt{
    int a[MAXSIZE],len;
    bool flag;
    BigInt() {
        len = 1;
        memset(a, 0, sizeof(a));
        flag = 0;
    }
    BigInt (const int b) {
        int c , d = b;
        len = 0;
        memset(a, 0, sizeof(a));
        if(!b) {
            len = 1;
            return ;
        }
        while(d) {
            a[len++] = d % (MAXN + 1);
            d /= (MAXN+1);
        }
    }
```

```
    BigInt(const char *s) {
        int t, k, index, l;
        memset(a, 0, sizeof(a));
        l = strlen(s);
        len = l/DLEN;
        if(l % DLEN) len ++;
        index = 0;
        for (int i = l - 1; i >= 0; i -= DLEN) {
            t = 0;
            k = i - DLEN + 1;
            if(k < 0) k = 0;
            for (int j = k; j <= i; j ++) t = t * 10 + s[
                 j] - '0';
            a[index ++] = t;
        }
    }
    BigInt (const BigInt& T) {
        memset(a, 0, sizeof(a));
        len = T.len;
        for (int i = 0; i < len; i ++) a[i] = T.a[i];
    }
    bool operator < (const BigInt &T) const {
        int ln;
        if(len < T.len) return 233;
        if(len == T.len) {
            ln = len - 1;
            while(ln >= 0 && a[ln] == T.a[ln]) --- ln;
            if(ln >= 0 && a[ln] < T.a[ln]) return 233;
            return 0;
        }
        return 0;
    }
    inline bool operator < (const int &t) const {
        BigInt tee(t);
        return *this < tee;
    }
    BigInt& operator = (const BigInt &T) {
        memset(a, 0, sizeof(a));
        len = T.len;
        for (int i = 0; i < len; i ++) a[i] = T.a[i];
        return *this;
    }
    BigInt operator + (const BigInt &T) const {
        BigInt t(*this);
        int big = len;
        if(T.len > len) big = T.len;
        for (int i = 0; i < big; i ++) {
            t.a[i] += T.a[i];
            if(t.a[i] > MAXN) {
                ++t.a[i + 1];
                t.a[i] -= MAXN + 1;
            }
        }
        if(t.a[big]) t.len = big + 1;
        else t.len = big;
        return t;
    }
    BigInt operator - (const BigInt &T) const {
        int big;
        bool ctf;
        BigInt t1, t2;
        if(*this < T) {
            t1 = T;
            t2 = *this;
            ctf = 1;
        }else {
            t1 = *this;
            t2 = T;
            ctf = 0;
        }
        big = t1.len;
        int j = 0;
        for (int i = 0; i < big; i ++) {
            if(t1.a[i] < t2.a[i]) {
                j = i + 1;
                while(t1.a[j] == 0) ++j;
                --- t1.a[j---];
                while(j > i) t1.a[j ---] += MAXN;
                t1.a[i] += MAXN + 1 - t2.a[i];
            }else t1.a[i] -= t2.a[i];
        }
        t1.len = big;
        while(t1.len > 1 && t1.a[t1.len - 1] == 0) {
            --- t1.len;
            --- big;
        }
        if(ctf) t1.a[big - 1] = -t1.a[big - 1];
        return t1;
    }
    BigInt operator * (const BigInt &T) const {
        BigInt res;
        int up;
        int te, tee;
        for (int i = 0; i < len; i ++) {
            up = 0;
            for (int j = 0; j < T.len; j ++) {
                te = a[i] * T.a[j] + res.a[i + j] + up;
                if(te > MAXN) {
                    tee = te - te / (MAXN + 1) * (MAXN +
                         1);
                    up = te / (MAXN + 1);
                    res.a[i + j] = tee;
                }else {
                    up = 0;
                    res.a[i + j] = te;
                }
            }
            if(up) res.a[i + T.len] = up;
```

```
        }
        res.len = len + T.len;
        while(res.len > 1 && res.a[res.len - 1] == 0) --
            res.len;
        return res;
    }
    BigInt operator / (const int &b) {
        BigInt res;
        int sum = 0, newlen = 0;
        for (int i = len-1; i >= 0; i --) {
            sum = sum * (MAXN+1) + a[i];
            if(sum < b) res.a[i] = 0;
            else {
                if(!newlen) newlen = i + 1;
                res.a[i] = sum / b;
                sum %= b;
            }
        }
        res.len = max(newlen, 1);
        return res;
    }
    int operator % (const int &b) const {
        int d = 0;
        for (int i = len - 1; i >= 0; i --)
            d = (d * (MAXN + 1) % b + a[i]) % b;
        return d;
    }
    BigInt operator ^ (const int &n) const {
        BigInt t(n), res(1);
        int y = n;
        while(y) {
            if(y & 1) res = res * t;
            t = t * t;
            y >>= 1;
        }
        return res;
    }
    inline void print() {
        printf("%d", a[len - 1]);
        for (int i = len - 2; i >= 0; i --)
            printf("%03d", a[i]);
        printf("\n");
    }
};
```

2.23 BSGS

```
map<long long, long long> Hash;
long long Mul(long long a, long long b, long long p) {
    long long L = a * (b >> 25LL) % p * (1LL << 25) % p;
    long long R = a * (b & ((1LL << 25) - 1)) % p;
    return (L + R) % p;
}
long long Pow(long long a, long long b, long long p) {
    a %= p;
    long long res = 1;
    while(b) {
        if(b & 1) res = Mul(res, a, p);
        a = Mul(a, a, p);
        b >>= 1;
    }
    return res;
}
/*
get ans for a^ans = b % p
A^{iS-j} =B mod p  A^{iS} = B*A^{j} mod p
A^{iS+j} = B mod p
*/
long long BSGS(long long a, long long b, long long p) {
    long long m = sqrt(p) + 1;
    long long res = 1;
    for (int j = 0; j <= m; j ++) {
        Hash[Mul(b, res, p)] = j;
        res = Mul(res, a, p);
    }
    for (int i = 1; i <= m; i ++) {
        long long k = Pow(a, i * m, p);
        if(Hash.count(k))
            return i * m - Hash[k];
    }
}
```

2.24 BM

```
//BM模板
//a[n] = f[1]a[n-1] + f[2]a[n-2] + f[3]a[n-3] + ...+ f[k]
    a[n-k]
//有限项
const long long mod = 998244353;
#define sz(x) ((int)(x).size())
typedef vector<long long> VI;
long long Ksm(long long a, long long b) {
    long long res = 1; a %= mod;
    assert(b >= 0);
    while(b) {
        if(b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
int _, n;
```

```
namespace Linear_Seq{
    const int N = 10010;
    long long res[N], base[N], _c[N], _md[N];
    vector<int> Md;
    void Mul(long long *a, long long *b, int k) {
        for (int i = 0; i < k+k; i ++) _c[i] = 0;
        for (int i = 0; i < k; i ++)
            if(a[i]) for (int j = 0; j < k; j ++)
                _c[i+j] = (_c[i+j] + a[i]*b[j]) % mod;
        for (int i = k + k - 1; i >= k; i --)
            if(_c[i]) for (int j = 0; j < sz(Md); j ++)
                _c[i-k+Md[j]] = (_c[i-k+Md[j]] - _c[i] *
                     _md[Md[j]]) % mod;
        for (int i = 0; i < k; i ++)
            a[i] = _c[i];
    }
    int solve(long long n, VI a, VI b) {
        long long ans = 0, pnt = 0;
        int k = sz(a);
        assert(sz(a) == sz(b));
        for (int i = 0; i < k; i ++) _md[k-1-i] = -a[i];
        _md[k] = 1; Md.clear();
        for (int i = 0; i < k; i ++)
            if(_md[i]) Md.push_back(i);
        for (int i = 0; i < k; i ++) res[i] = base[i] =
             0;
        res[0] = 1;
        while((1ll<<pnt) <= n) pnt ++;
        for (int p = pnt; p >= 0; p --) {
            Mul(res, res, k);
            if((n>>p) & 1) {
                for (int i = k-1; i >= 0; i --) res[i+1]
                     = res[i];
                res[0] = 0;
                for (int j = 0; j < sz(Md); j ++)
                    res[Md[j]] = (res[Md[j]] - res[k] *
                         _md[Md[j]]) % mod;
            }
        }
        for (int i = 0; i < k; i ++) ans = (ans + res[i]
             * b[i]) % mod;
        if(ans < 0) ans += mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1, 1), B(1, 1);
        int L = 0, m = 1, b = 1;
        for (int n = 0; n < sz(s); n ++) {
            long long d = 0;
            for (int i = 0; i < L + 1; i ++) d = (d + (
                 long long)C[i] * s[n-i]) % mod;
            if (d == 0) ++m;
            else if(2 * L <= n) {
                VI T = C;
                long long c = mod - d * Ksm(b, mod-2) %
                     mod;
                while(sz(C) < sz(B) + m) C.push_back(0);
                for (int i = 0; i < sz(B); i ++) C[i+m] =
                      (C[i+m] + c * B[i]) % mod;
                L = n + 1 - L; B = T;
                b = d; m = 1;
            }else {
                long long c = mod - d * Ksm(b, mod-2) %
                     mod;
                while(sz(C) < sz(B) + m) C.push_back(0);
                for (int i = 0; i < sz(B); i ++) C[i+m] =
                      (C[i+m] + c * B[i]) % mod;
                ++ m;
            }
        }
        return C;
    }
    int Gao(VI a, long long n) { //得到第n项
        VI c = BM(a);
        c.erase(c.begin());
        for (int i = 0; i < sz(c); i ++) c[i] = (mod-c[i
             ]) % mod;
        return solve(n, c, VI(a.begin(), a.begin()+sz(c))
             );
    }
};
using namespace Linear_Seq;
void solve() { //预处理前3k项
    long long n, k;
    scanf("%lld %lld", &n, &k);
    VI v, f;
    f.push_back(0);
    for (int i = 0; i < k; i ++) { //f只有k项
        long long x;
        scanf("%lld", &x);
        f.push_back(x);
    }
    for (int i = 0; i < k; i ++) { //a的前k项
        long long x;
        scanf("%lld", &x);
        v.push_back(x);
    }
    for (int i = k; i <= 2 * k; i ++) {//a的前3k项
        long long x = 0;
        for (int j = 1; j <= k; j ++)
            x = (x + f[j] * v[i-j]) % mod;
        v.push_back(x);
    }
    printf("%lld\n", Gao(v, n));
}
int main() {
    solve();
    return 0;
```

```
}
//另一个板子
#define maxk 100005
#define maxn 200005
const int mod = 998244353;
#define mul(x, y) static_cast<long long> (x) * (y) % mod
namespace Math {
        inline int pw(int base, int p) {
                static int res; res = 1;
        while(p) {
            if(p & 1) res = mul(res, base);
            base = mul(base, base);
            p >>= 1;
        }
                return res;
        }
        inline int inv(int x) { return pw(x, mod - 2); }
}
inline void reduce(int &x) { x += x >> 31 & mod; }
namespace Poly {
#define N maxn
        int lim, s, rev[N], Wn[N];
        inline void init(const int n) { //初始化
                lim = 1, s = -1;
        while (lim < n) lim <<= 1, ++s;
                for (register int i = 1; i < lim; ++i)
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << s;
                const int t = Math::pw(3, (mod - 1) / lim
                     ); *Wn = 1;
        for (register int *i = Wn + 1; i != Wn + lim; ++i
             )
            *i = mul(*(i - 1), t);
        }
        inline void FFT(int *A, const int op = 1) { //FFT
                for (register int i = 1; i < lim; ++i)
            if (i < rev[i]) std::swap(A[i], A[rev[i]]);
                for (register int mid = 1; mid < lim; mid
                      <<= 1) {
                  const int t = lim / mid >> 1;
                  for (register int i = 0; i < lim;
                       i += mid << 1)
                    for (register int j = 0;
                         j < mid; ++j) {
                       const int X = A[i
                            + j], Y =
                           mul(A[i + j
                           + mid], Wn[t
                            * j]);
                       reduce(A[i + j]
                            += Y - mod),
                            reduce(A[i
                           + j + mid] =
                            X - Y);
                    }
                }
        if (!op) {
                const int ilim = Math::inv(lim);
                for (register int *i = A; i != A
                     + lim; ++i) *i = mul(*i,
                     ilim);
                std::reverse(A + 1, A + lim);
        }
    }
    void INV(int *A, int *B, int n) {  //多项式A求逆
         到B, [0,n-1]
        if (n == 1) { *B = Math::inv(*A); return
             ; }
        static int C[N], D[N];
        const int len = n + 1 >> 1;
        INV(A, B, len), init(len * 3);
        std::memcpy(C, A, n << 2), std::memset(C
             + n, 0, lim - n << 2);
        std::memcpy(D, B, len << 2), std::memset(
             D + len, 0, lim - len << 2);
        FFT(C), FFT(D);
        for (int i = 0; i < lim; ++i) D[i] = (2 -
              mul(D[i], C[i]) + mod) * D[i] % mod
             ;
        FFT(D, 0);
        std::memcpy(B + len, D + len, n - len <<
             2);
    }
    int G[N], INVG[N];
    void DIV(int *A, int *Q, int n, int m) {
        static int C[N];
        const int len = n - m + 1;
        std::reverse_copy(A, A + n, C), std::
             memset(C + len, 0, lim - len << 2);
        FFT(C);
        for (int i = 0; i < lim; ++i) Q[i] = mul(
             C[i], INVG[i]);
        FFT(Q, 0), std::reverse(Q, Q + len);
    }
    void DIV_MOD(int *A, int *R, int n, int m) {
        static int Q[N];
        const int len = n - m + 1;
        DIV(A, Q, n, m), std::memset(Q + len, 0,
             lim - len << 2);
        FFT(Q);
        for (int i = 0; i < lim; ++i) R[i] = mul(
             G[i], Q[i]);
        FFT(R, 0);
        for (int i = 0; i < m; ++i) reduce(R[i] =
              A[i] - R[i]);
    }
    void POW(int *A, int p, int m) {
        if (!p) return ;
        POW(A, p >> 1, m);
```

```
                static int T[N];
                std::memcpy(T, A, m << 2), std::memset(T
                    + m, 0, lim - m << 2);
                FFT(T);
                for (int i = 0; i < lim; ++i) T[i] = mul(
                    T[i], T[i]);
                FFT(T, 0);
                if (p & 1) {
                        for (int i = 2 * m - 1; ~i; --i)
                            T[i] = T[i - 1];
                        T[0] = 0;
                }
                DIV_MOD(T, A, 2 * m, m + 1);
        }
        int solve(int *f, int *a, int n, int k) { //a为递
            推式0~k-1项, f为转移数组1~k项
                static int A[maxn], B[maxn];
                for (int i = 1; i <= k; ++i) reduce(G[k -
                    i] = -f[i]);
                G[k] = A[0] = 1;
                std::reverse_copy(G, G + k + 1, B), B[k]
                    = 0;
                INV(B, INVG, k), init(k << 1);
                FFT(G), FFT(INVG);
                Poly::POW(A, n, k);
                int ans = 0;
                for (int i = 0; i < k; ++i) reduce(ans +=
                    mul(A[i], a[i]) - mod);
                return ans;
        }
#undef N
}
int n, k;
int f[maxk], a[maxk];
int main() {
        /* 能求线性递推和mod 998244353的多项是求逆, 其他
            的好
        像可以求, 但是不会, 先打个板子, 以后再说把*/
    // int n;
    // scanf("%d", &n);
    // for (int i = 0; i < n; i ++)
    //     scanf("%d", &a[i]);
    // Poly::INV(a, f, n);
    // for (int i = 0; i < n; i ++)
    //     printf("%d ", f[i]);
    // printf("\n");
    // a(n)=f(i)*a(n-i) {1<=i<=k}
        std::ios::sync_with_stdio(false), std::cin.tie(0)
            , std::cout.tie(0);
        std::cin >> n >> k;
        for (int i = 1; i <= k; ++i) std::cin >> f[i];
        for (int i = 0; i < k; ++i) std::cin >> a[i],
            reduce(a[i]);
        std::cout << Poly::solve(f, a, n, k) << '\n';
        return 0;
}
```

3 数据结构

3.1 线段树套伸展树

```
/* BZOJ 3196（线段树套伸展树）
1.查询k在区间内的排名
2.查询区间内排名为k的值
3.修改某一位值上的数值
4.查询k在区间内的前驱(前驱定义为小于x，且最大的数)
5.查询k在区间内的后继(后继定义为大于x，且最小的数) */
#include <bits/stdc++.h>
const int inf = 2147483647;
const int maxn = 5e4 + 5;
const int maxm = maxn * 25;
int n;
int arr[maxn];
namespace SplayTree {
  int rt[maxm], tot;
  int fa[maxm], son[maxm][2];
  int val[maxm], cnt[maxm];
  int sz[maxm];
  void Push(int o) {
    sz[o] = sz[son[o][0]] + sz[son[o][1]] + cnt[o];
  }
  bool Get(int o) {
    return o == son[fa[o]][1];
  }
  void Clear(int o) {
    son[o][0] = son[o][1] = fa[o] = val[o] = sz[o] = cnt[
        o] = 0;
  }
  void Rotate(int o) {
    int p = fa[o], q = fa[p], ck = Get(o);
    son[p][ck] = son[o][ck ^ 1];
    fa[son[o][ck ^ 1]] = p;
    son[o][ck ^ 1] = p;
    fa[p] = o; fa[o] = q;
    if (q) son[q][p == son[q][1]] = o;
    Push(p); Push(o);
  }
  void Splay(int &root, int o) {
    for (int f = fa[o]; (f = fa[o]); Rotate(o))
      if (fa[f]) Rotate(Get(o) == Get(f) ? f : o);
    root = o;
  }
  void Insert(int &root, int x) {
    if (!root) {
      val[++tot] = x;
      cnt[tot]++;
      root = tot;
      Push(root);
      return;
    }
    int cur = root, f = 0;
    while (true) {
      if (val[cur] == x) {
        cnt[cur]++;
        Push(cur); Push(f);
        Splay(root, cur);
        break;
      }
      f = cur;
      cur = son[cur][val[cur] < x];
      if (!cur) {
        val[++tot] = x;
        cnt[tot]++;
        fa[tot] = f;
        son[f][val[f] < x] = tot;
        Push(tot); Push(f);
        Splay(root, tot);
        break;
      }
    }
  }
  int GetRank(int &root, int x) {
    int ans = 0, cur = root;
    while (cur) {
      if (x < val[cur]) {
        cur = son[cur][0];
        continue;
      }
      ans += sz[son[cur][0]];
      if (x == val[cur]) {
        Splay(root, cur);
        return ans;
      }
      if (x > val[cur]) {
        ans += cnt[cur];
        cur = son[cur][1];
      }
    }
    return ans;
  }
  int GetKth(int &root, int k) {
    int cur = root;
    while (true) {
      if (son[cur][0] && k <= sz[son[cur][0]]) cur = son[
          cur][0];
      else {
        k -= cnt[cur] + sz[son[cur][0]];
        if (k <= 0) return cur;
        cur = son[cur][1];
      }
    }
  }
  int Find(int &root, int x) {
```

```
    int ans = 0, cur = root;
    while (cur) {
      if (x < val[cur]) {
        cur = son[cur][0];
        continue;
      }
      ans += sz[son[cur][0]];
      if (x == val[cur]) {
        Splay(root, cur);
        return ans + 1;
      }
      ans += cnt[cur];
      cur = son[cur][1];
    }
  }
  int GetPrev(int &root) {
    int cur = son[root][0];
    while (son[cur][1]) cur = son[cur][1];
    return cur;
  }
  int GetPrevVal(int &root, int x) {
    int ans = -inf, cur = root;
    while (cur) {
      if (x > val[cur]) {
        ans = std::max(ans, val[cur]);
        cur = son[cur][1];
        continue;
      }
      cur = son[cur][0];
    }
    return ans;
  }
  int GetNext(int &root) {
    int cur = son[root][1];
    while (son[cur][0]) cur = son[cur][0];
    return cur;
  }
  int GetNextVal(int &root, int x) {
    int ans = inf, cur = root;
    while (cur) {
      if (x < val[cur]) {
        ans = std::min(ans, val[cur]);
        cur = son[cur][0];
        continue;
      }
      cur = son[cur][1];
    }
    return ans;
  }
  void Delete(int &root, int x) {
    Find(root, x);
    if (cnt[root] > 1) {
      cnt[root]--;
      Push(root);
      return;
    }
    if (!son[root][0] && !son[root][1]) {
      Clear(root);
      root = 0;
      return;
    }
    if (!son[root][0]) {
      int cur = root;
      root = son[root][1];
      fa[root] = 0;
      Clear(cur);
      return;
    }
    if (!son[root][1]) {
      int cur = root;
      root = son[root][0];
      fa[root] = 0;
      Clear(cur);
      return;
    }
    int p = GetPrev(root), cur = root;
    Splay(root, p);
    fa[son[cur][1]] = p;
    son[p][1] = son[cur][1];
    Clear(cur);
    Push(root);
  }
};

namespace SegTree {
  int tree[maxn * 4];
  void Build(int o, int l, int r) {
    for (int i = l; i <= r; ++i) SplayTree::Insert(tree[o
        ], arr[i - 1]);
    if (l == r) return;
    int m = (l + r) / 2;
    Build(o * 2, l, m);
    Build(o * 2 + 1, m + 1, r);
  }
  void Modify(int o, int l, int r, int ll, int rr, int u,
      int v) {
    SplayTree::Delete(tree[o], u); SplayTree::Insert(tree
        [o], v);
    if (l == r) return;
    int m = (l + r) / 2;
    if (ll <= m) Modify(o * 2, l, m, ll, rr, u, v);
    if (rr > m) Modify(o * 2 + 1, m + 1, r, ll, rr, u, v)
        ;
  }
  int QueryRank(int o, int l, int r, int ll, int rr, int
      v) {
    if (ll <= l && rr >= r) return SplayTree::GetRank(
        tree[o], v);
```

```
    int m = (l + r) / 2, ans = 0;
    if (ll <= m) ans += QueryRank(o * 2, l, m, ll, rr, v)
        ;
    if (rr > m) ans += QueryRank(o * 2 + 1, m + 1, r, ll,
         rr, v);
    return ans;
  }
  int QueryPrev(int o, int l, int r, int ll, int rr, int
      v) {
    if (ll <= l && rr >= r) return SplayTree::GetPrevVal(
        tree[o], v);
    int m = (l + r) / 2, ans = -inf;
    if (ll <= m) ans = std::max(ans, QueryPrev(o * 2, l,
        m, ll, rr, v));
    if (rr > m) ans = std::max(ans, QueryPrev(o * 2 + 1,
        m + 1, r, ll, rr, v));
    return ans;
  }
  int QueryNext(int o, int l, int r, int ll, int rr, int
      v) {
    if (ll <= l && rr >= r) return SplayTree::GetNextVal(
        tree[o], v);
    int m = (l + r) / 2, ans = inf;
    if (ll <= m) ans = std::min(ans, QueryNext(o * 2, l,
        m, ll, rr, v));
    if (rr > m) ans = std::min(ans, QueryNext(o * 2 + 1,
        m + 1, r, ll, rr, v));
    return ans;
  }
  int QueryKth(int ll, int rr, int v) {
    int l = 0, r = 1e8 + 10;
    while (l < r) {
      int m = ((l + r) / 2) + 1;
      if (QueryRank(1, 1, n, ll, rr, m) < v) l = m;
      else r = m - 1;
    }
    return l;
  }
};
int main() {
  std::ios::sync_with_stdio(false);
  std::cout.tie(0);
  std::cin.tie(0);
  int m;
  std::cin >> n >> m;
  for (int i = 0; i < n; ++i) std::cin >> arr[i];
  SplayTree::tot = 0;
  SegTree::Build(1, 1, n);
  for (int i = 0, op, l, r, pos, k; i < m; ++i) {
    std::cin >> op;
    if (op == 1) {
      std::cin >> l >> r >> k;
      std::cout << SegTree::QueryRank(1, 1, n, l, r, k) +
          1 << '\n';
    }
    else if (op == 2) {
      std::cin >> l >> r >> k;
      std::cout << SegTree::QueryKth(l, r, k) << '\n';
    }
    else if (op == 3) {
      std::cin >> pos >> k;
      SegTree::Modify(1, 1, n, pos, pos, arr[pos - 1], k)
          ;
      arr[pos - 1] = k;
    }
    else if (op == 4) {
      std::cin >> l >> r >> k;
      std::cout << SegTree::QueryPrev(1, 1, n, l, r, k)
          << '\n';
    }
    else if (op == 5) {
      std::cin >> l >> r >> k;
      std::cout << SegTree::QueryNext(1, 1, n, l, r, k)
          << '\n';
    }
  }
  return 0;
}
```

3.2 线段树

3.2.1 线段树合并

```
// BZOJ2212: 交换左右子树后最小逆序对
#include <bits/stdc++.h>
const int maxn = 1e7 + 5;
template <typename t>
inline bool Read(t &ret) {
  char c; int sgn;
  if (c = getchar(), c == EOF) return false;
  while (c != '-' && (c < '0' || c > '9')) c = getchar();
  sgn = (c == '-') ? -1 : 1;
  ret = (c == '-') ? 0 : (c - '0');
  while (c = getchar(), c >= '0' && c <= '9') ret = ret *
      10 + (c - '0');
  ret *= sgn;
  return true;
}
struct node {
  int sz, lson, rson;
  node() { sz = lson = rson = 0; }
};
int n;
int tot;
node tree[maxn];
```

```
long long ans1, ans2;
long long ans;
int Build(int l, int r, int c) {
  tree[++tot].sz = 1;
  if (l == r) return tot;
  int m = (l + r) / 2, o = tot;
  if (c <= m) tree[o].lson = Build(l, m, c);
  else tree[o].rson = Build(m + 1, r, c);
  return o;
}
int Merge(int l, int r, int x, int y) {
  if (!x || !y) return x + y;
  if (l == r) {
    tree[++tot].sz = tree[x].sz + tree[y].sz;
    return tot;
  }
  ans1 += 1ll * tree[tree[x].rson].sz * tree[tree[y].lson
       ].sz;
  ans2 += 1ll * tree[tree[x].lson].sz * tree[tree[y].rson
       ].sz;
  int m = (l + r) / 2, o = ++tot;
  tree[o].lson = Merge(l, m, tree[x].lson, tree[y].lson);
  tree[o].rson = Merge(m + 1, r, tree[x].rson, tree[y].
       rson);
  tree[o].sz = tree[x].sz + tree[y].sz;
  return o;
}
int Dfs() {
  int c = 0;
  Read(c);
  if (c) return Build(1, n, c);
  int o = Merge(1, n, Dfs(), Dfs());
  ans += std::min(ans1, ans2);
  ans1 = ans2 = 0;
  return o;
}
int main() {
  Read(n);
  Dfs();
  printf("%lld", ans);
  return 0;
}
```

3.2.2 线段树

```
const int maxn = "Edit";
struct SegTree {
  int n;
  long long sum[maxn * 4], lazy[maxn * 4];
  long long Unite(const long long &k1, const long long &
       k2) {
    return k1 + k2;
  }
  void Pull(int o) {
    sum[o] = Unite(sum[o * 2], sum[o * 2 + 1]);
  }
  void Push(int o, int l, int r) {
    int m = (l + r) / 2;
    if (lazy[o] != 0) {
      sum[o * 2] += (m - l + 1) * lazy[o];
      sum[o * 2 + 1] += (r - m) * lazy[o];
      lazy[o * 2] += lazy[o];
      lazy[o * 2 + 1] += lazy[o];
      lazy[o] = 0;
    }
  }
  void Build(int o, int l, int r, long long arr[]) {
    sum[o] = lazy[o] = 0;
    if (l == r) {
      sum[o] = arr[l];
      return;
    }
    int m = (l + r) / 2;
    Build(o * 2, l, m, arr);
    Build(o * 2 + 1, m + 1, r, arr);
    Pull(o);
  }
  void Init(int _n, long long arr[]) {
    n = _n;
    Build(1, 1, n, arr);
  }
  void Modify(int o, int l, int r, int ll, int rr, long
       long v) {
    if (ll <= l && rr >= r) {
      sum[o] += (r - l + 1) * v;
      lazy[o] += v;
      return;
    }
    Push(o, l, r);
    int m = (l + r) / 2;
    if (ll <= m) Modify(o * 2, l, m, ll, rr, v);
    if (rr > m) Modify(o * 2 + 1, m + 1, r, ll, rr, v);
    Pull(o);
  }
  void Modify(int ll, int rr, long long v) {
    Modify(1, 1, n, ll, rr, v);
  }
  long long Query(int o, int l, int r, int ll, int rr) {
    if (ll <= l && rr >= r) return sum[o];
    Push(o, l, r);
    int m = (l + r) / 2;
    long long ret = 0;
    if (ll <= m) ret = Unite(ret, Query(o * 2, l, m, ll,
         rr));
    if (rr > m) ret = Unite(ret, Query(o * 2 + 1, m + 1,
         r, ll, rr));
```

```
    return ret;
  }
  long long Query(int ll, int rr) {
    return Query(1, 1, n, ll, rr);
  }
};
```

3.2.3 矩形面积异或并

```
// CodeForces GYM 101982 F 矩形面积异或并
#include <bits/stdc++.h>
std::vector<int> x;
int Get(int k) {
  return std::lower_bound(x.begin(), x.end(), k) - x.
       begin();
}
struct SegTree {
  struct Node {
    int v, lazy;
    Node() { v = lazy = 0; }
  };
  int n;
  std::vector<Node> tree;
  Node Unite(const Node &k1, const Node &k2) {
    Node ans;
    ans.v = k1.v + k2.v;
    return ans;
  }
  void Pull(int o) {
    tree[o] = Unite(tree[o * 2], tree[o * 2 + 1]);
  }
  void Push(int o, int l, int r) {
    int m = (l + r) / 2;
    if (tree[o].lazy != 0) {
      tree[o * 2].v = x[m] - x[l - 1] - tree[o * 2].v;
      tree[o * 2 + 1].v = x[r] - x[m] - tree[o * 2 + 1].v
           ;
      tree[o * 2].lazy ^= 1;
      tree[o * 2 + 1].lazy ^= 1;
      tree[o].lazy = 0;
    }
  }
  void Build(int o, int l, int r) {
    if (l == r) return;
    int m = (l + r) / 2;
    Build(o * 2, l, m);
    Build(o * 2 + 1, m + 1, r);
    Pull(o);
  }
  SegTree(int _n): n(_n) {
    tree.resize(n << 2);
    Build(1, 1, n);
  }
  void Modify(int o, int l, int r, int ll, int rr) {
    if (ll <= l && rr >= r) {
      tree[o].v = x[r] - x[l - 1] - tree[o].v;
      tree[o].lazy ^= 1;
      return;
    }
    Push(o, l, r);
    int m = (l + r) / 2;
    if (ll <= m) Modify(o * 2, l, m, ll, rr);
    if (rr > m) Modify(o * 2 + 1, m + 1, r, ll, rr);
    Pull(o);
  }
  void Modify(int ll, int rr) {
    Modify(1, 1, n, ll, rr);
  }
  Node Query(int o, int l, int r, int ll, int rr) {
    if (ll <= l && rr >= r) return tree[o];
    Push(o, l, r);
    int m = (l + r) / 2;
    Node ans;
    if (ll <= m) ans = Unite(ans, Query(o * 2, l, m, ll,
         rr));
    if (rr > m) ans = Unite(ans, Query(o * 2 + 1, m + 1,
         r, ll, rr));
    Pull(o);
    return ans;
  }
  Node Query() {
    return Query(1, 1, n, 1, n);
  }
};
struct seg { int l, r, h, flag; };
bool operator < (seg k1, seg k2) { return k1.h < k2.h; }
std::vector<seg> s;
int main() {
  std::ios::sync_with_stdio(false);
  std::cout.tie(nullptr);
  std::cin.tie(nullptr);
  int n; std::cin >> n;
  for (int i = 0, x1, y1, x2, y2; i < n; ++i) {
    std::cin >> x1 >> y1 >> x2 >> y2;
    if (x1 > x2) std::swap(x1, x2);
    if (y1 > y2) std::swap(y1, y2);
    x.emplace_back(x1); x.emplace_back(x2);
    s.emplace_back((seg){x1, x2, y1, 1});
    s.emplace_back((seg){x1, x2, y2, -1});
  }
  sort(s.begin(), s.end());
  sort(x.begin(), x.end());
  x.erase(unique(x.begin(), x.end()), x.end());
  SegTree tree((int)x.size());
  long long ans = 0;
  for (int i = 0, l, r; i < (int)s.size() - 1; ++i) {
```

```
    l = Get(s[i].l), r = Get(s[i].r);
    tree.Modify(l + 1, r);
    ans += (long long)tree.Query().v * (s[i + 1].h - s[i
        ].h);
  }
  std::cout << ans << '\n';
  return 0;
}
```

3.2.4 矩形面积并

```
// HDU 1542 矩形面积并
#include <bits/stdc++.h>
typedef double db;
const int maxn = 1e2 + 5;
const db eps = 1e-9;
int Sgn(db k) {
  return std::fabs(k) < eps ? 0 : (k < 0 ? -1 : 1);
}
int Cmp(db k1, db k2) {
  return Sgn(k1 - k2);
}
struct Seg {
  db l, r, h;
  int flag;
};
bool operator < (Seg &k1, Seg &k2) {
  return Cmp(k1.h, k2.h) < 0;
}
std::vector<Seg> Segs;
std::vector<db> pos;
int BinarySearch(db k) {
  int ret = (int)pos.size() - 1, l = 0, r = (int)pos.size
      () - 1;
  while (l <= r) {
    int m = (l + r) >> 1;
    if (Cmp(pos[m], k) >= 0) {
      ret = m;
      r = m - 1;
    }
    else l = m + 1;
  }
  return ret;
}
struct Node {
  int l, r, cnt;
  db len;
};
Node Seg_tree[maxn * 10];
void Pull(int o) {
  if (Seg_tree[o].cnt) Seg_tree[o].len = pos[Seg_tree[o].
      r + 1] - pos[Seg_tree[o].l];
  else if (Seg_tree[o].l == Seg_tree[o].r) Seg_tree[o].
      len = 0.0;
  else Seg_tree[o].len = Seg_tree[o << 1].len + Seg_tree[
      o << 1 | 1].len;
}
void Build(int l, int r, int o) {
  Seg_tree[o].l = l; Seg_tree[o].r = r;
  Seg_tree[o].cnt = 0; Seg_tree[o].len = 0.0;
  if (l == r) return;
  int Mid = (l + r) >> 1;
  Build(l, Mid, o << 1);
  Build(Mid + 1, r, o << 1 | 1);
  Pull(o);
}
void Update(int l, int r, int v, int o) {
  if (l <= Seg_tree[o].l && r >= Seg_tree[o].r) {
    Seg_tree[o].cnt += v;
    Pull(o);
    return;
  }
  int Mid = (Seg_tree[o].l + Seg_tree[o].r) >> 1;
  if (r <= Mid) Update(l, r, v, o << 1);
  else if (l > Mid) Update(l, r, v, o << 1 | 1);
  else {
    Update(l, Mid, v, o << 1);
    Update(Mid + 1, r, v, o << 1 | 1);
  }
  Pull(o);
}
int cas;
int n;
db x1, y1, x2, y2;
db ans;
int main() {
  while (~scanf("%d", &n) && n) {
    Segs.clear();
    pos.clear();
    for (int i = 0; i < n; ++i) {
      scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
      Segs.push_back((Seg){x1, x2, y1, 1});
      Segs.push_back((Seg){x1, x2, y2, -1});
      pos.push_back(x1);
      pos.push_back(x2);
    }
    std::sort(Segs.begin(), Segs.end());
    std::sort(pos.begin(), pos.end(), [&](db k1, db k2) {
        return Cmp(k1, k2) < 0; });
    int cur = 1;
    for (int i = 1; i < (int)pos.size(); ++i)
      if (Cmp(pos[i], pos[i - 1]) != 0)
        pos[cur++] = pos[i];
    pos.erase(pos.begin() + cur, pos.end());
    Build(0, (int)pos.size(), 1);
    ans = 0.0;
```

```
    for (int i = 0; i < (int)Segs.size() - 1; ++i) {
      int l = BinarySearch(Segs[i].l), r = BinarySearch(
          Segs[i].r);
      Update(l, r - 1, Segs[i].flag, 1);
      ans += (Segs[i + 1].h - Segs[i].h) * Seg_tree[1].
          len;
    }
    printf("Test case #%d\n", ++cas);
    printf("Total explored area: %.2lf\n\n", ans);
  }
  return 0;
}
```

3.3 树链剖分

```
const int maxn = "Edit";
int n;
long long val[maxn];
int fa[maxn], dep[maxn];
int sz[maxn], son[maxn];
int rk[maxn], top[maxn];
int id[maxn];
int dfs_clock;
std::vector<int> g[maxn];
void Dfs1(int u, int p, int d) {
  fa[u] = p;
  dep[u] = d;
  sz[u] = 1;
  for (int &v : g[u]) {
    if (v == p) continue;
    Dfs1(v, u, d + 1);
    sz[u] += sz[v];
    if (sz[v] > sz[son[u]]) son[u] = v;
  }
}
Dfs2(int u, int tp) {
  top[u] = tp;
  id[u] = ++dfs_clock;
  rk[dfs_clock] = u;
  if (!son[u]) return;
  Dfs2(son[u], tp);
  for (int &v : g[u]) {
    if (v == son[u] || v == fa[u]) continue;
    Dfs2(v, v);
  }
}
long long Modify(int u, int v, long long c) {
  while (top[u] != top[v]) {
    if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
    /* modify c from [id[top[u]], id[u]] in val */
    u = fa[top[u]];
  }
  if (id[u] > id[v]) std::swap(u, v);
  /* modify c from [id[u], id[v]] in val */
}
long long Query(int u, int v) {
  long long ret = 0;
  while (top[u] != top[v]) {
    if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
    ret += /* query from [id[top[u]], id[u]] in val */
    u = fa[top[u]];
  }
  if (id[u] > id[v]) std::swap(u, v);
  ret += /* query from [id[u], id[v]] in val */
  return ret;
}
```

3.4 树状数组

```
const int maxn = "Edit";
struct BitTree {
  int tree[maxn];
  void Init() {
    memset(tree, 0, sizeof(tree));
  }
  void Modify(int x, int v) {
    for (int i = x; i < maxn; i += i & (-i))
      tree[i] += v;
  }
  int Query(int x) {
    int ret = 0;
    for (int i = x; i > 0; i -= i & (-i))
      ret += tree[i];
    return ret;
  }
  int GetRank(int v) {
    int ret = 1;
    --v;
    for (int i = v; i > 0; i -= i & (-i))
      ret += tree[i];
    return ret;
  }
  int GetKth(int k) { // kth min
    int ret = 0, cnt = 0, max = log2(maxn);
    for (int i = max; i >= 0; --i) {
      ret += (1 << i);
      if (ret >= maxn || cnt += tree[ret] >= k) ret -= (1
          << i);
      else cnt += tree[ret];
    }
    return ++ret;
  }
```

```
  int GetPrev(int v) {
    return GetKth(GetRank(v) - 1);
  }
  int GetNext(int v) {
    return GetKth(GetRank(v) + 1);
  }
};
```

3.5 最近公共祖先

3.5.1 欧拉序 +RMQ

```
const int maxn = "Edit";
const int maxlog = "Edit";
int n;
std::vector<int> g[maxn];
int ele[maxn * 2], dep[maxn * 2];
int fi[maxn], fa[maxn];
int tot;
int dp[maxn * 2][maxlog];
void Dfs(int u, int p, int d) {
  ele[++tot] = u;
  fi[u] = tot;
  dep[tot] = d;
  fa[u] = p;
  for (int &v : g[u]) {
    if (v == p) continue;
    Dfs(v, u, d + 1);
    ele[++tot] = u;
    dep[tot] = d;
  }
}
void Init() {
  for(int i = 1; i <= 2 * n - 1; ++i) dp[i][0] = i;
  for (int j = 1; (1 << j) <= 2 * n - 1; ++j)
    for (int i = 1; i + (1 << j) - 1 <= 2 * n - 1; ++i)
      dp[i][j] = dep[dp[i][j - 1]] < dep[dp[i + (1 << j -
            1)][j - 1]] ? dp[i][j - 1] : dp[i + (1 << (j
          - 1))][j - 1];
}
int Query(int l, int r) {
  if (l > r) std::swap(l, r);
  int len = log2(r - l + 1);
  return dep[dp[l][len]] <= dep[dp[r - (1 << len) + 1][
      len]] ? dp[l][len] : dp[r - (1 << len) + 1][len];
}
int GetLCA(int u, int v) {
  return ele[Query(fi[u], fi[v])];
}
```

3.5.2 倍增

```
const int maxn = "Edit";
const int maxlog = "Edit";
int n, k; // k = log2(n) + 1
std::vecotr<int> g[maxn];
int anc[maxn][maxlog];
int dep[maxn];
// 从根节点开始深搜预处理
void Dfs(int u, int p, int d) {
  anc[u][0] = p;
  dep[u] = d;
  for (int &v : g[u]) {
    if (v == p) continue;
    Dfs(v, u, d + 1);
  }
}
void Swim(int &u, int h) {
  for (int i = 0; h > 0; ++i) {
    if (h & 1) u = anc[u][i];
    h >>= 1;
  }
}
int GetLCA(int u, int v) {
  if (dep[u] < dep[v]) std::swap(u, v);
  Swim(u, dep[u] - dep[v]);
  if (u == v) return v;
  for (int i = k - 1; i >= 0; --i) {
    if (anc[u][i] != anc[v][i]) {
      u = anc[u][i];
      v = anc[v][i];
    }
  }
  return anc[u][0];
}
```

3.5.3 tarjan

```
const int maxn = "Edit";
const int maxm = "Edit";
int n;
int pre[maxn];
int Find(int o) {
  return pre[o] == o ? o : pre[o] = Find(pre[o]);
}
void Union(int u, int v) {
  if (Find(u) != Find(v)) pre[Find(u)] = Find(v);
}
std::vector<int> g[maxn];
bool vis[maxn];
struct query { int v, id; };
```

```
std::vector<query> qry[maxm];
void Init() {
  for (int i = 1; i <= n; ++i) {
    pre[i] = i;
    vis[i] = false;
  }
}
void Tarjan(int u) {
  vis[u] = true;
  for (int &v : g[u]) {
    if (vis[v]) continue;
    Tarjan(v);
    Union(v, u);
  }
  for (query &q : qry[u]) {
    if (vis[q.v]) ans[q.id] = Find(q.v);
  }
}
```

3.6 伸展树

```
const int inf = "Edit"
const int maxn = "Edit";
struct SplayTree {
  int rt, tot;
  int fa[maxn], son[maxn][2];
  int val[maxn], cnt[maxn];
  int sz[maxn];
  bool lazy[maxn];
  void Pull(int o) {
    sz[o] = sz[son[o][0]] + sz[son[o][1]] + cnt[o];
  }
  void Push(int o) {
    if (lazy[o]) {
      std::swap(son[o][0], son[o][1]);
      if (son[o][0]) lazy[son[o][0]] ^= 1;
      if (son[o][1]) lazy[son[o][1]] ^= 1;
      lazy[o] = 0;
    }
  }
  bool Get(int o) {
    return o == son[fa[o]][1];
  }
  void Clear(int o) {
    son[o][0] = son[o][1] = fa[o] = val[o] = sz[o] = cnt[
        o] = 0;
  }
  void Rotate(int o) {
    int p = fa[o], q = fa[p], ck = Get(o);
    son[p][ck] = son[o][ck ^ 1];
    fa[son[o][ck ^ 1]] = p;
    son[o][ck ^ 1] = p;
    fa[p] = o; fa[o] = q;
    if (q) son[q][p == son[q][1]] = o;
    Pull(p); Pull(o);
  }
  void Splay(int o) {
    for (int f = fa[o]; f = fa[o], f; Rotate(o))
      if (fa[f]) Rotate(Get(o) == Get(f) ? f : o);
    rt = o;
  }
  // 旋转o节点到节点tar
  void Splay(int o, int tar = 0) {
    for (int f = fa[o]; (f = fa[o]) != tar; Rotate(o)) {
      Pull(fa[f]); Pull(f); Pull(o);
      if (fa[f] != tar) {
        if (Get(o) == Get(f)) Rotate(f);
        else Rotate(o);
      }
    }
    if (!tar) rt = o;
  }
  void Insert(int x) {
    if (!rt) {
      val[++tot] = x;
      cnt[tot]++;
      rt = tot;
      Pull(rt);
      return;
    }
    int cur = rt, f = 0;
    while (true) {
      if (val[cur] == x) {
        cnt[cur]++;
        Pull(cur); Pull(f);
        Splay(cur);
        break;
      }
      f = cur;
      cur = son[cur][val[cur] < x];
      if (!cur) {
        val[++tot] = x;
        cnt[tot]++;
        fa[tot] = f;
        son[f][val[f] < x] = tot;
        Pull(tot); Pull(f);
        Splay(tot);
        break;
      }
    }
  }
  int GetRank(int x) {
    int ans = 0, cur = rt;
    while (true) {
      if (x < val[cur]) cur = son[cur][0];
```

```
    else {
      ans += sz[son[cur][0]];
      if (x == val[cur]) {
        Splay(cur);
        return ans + 1;
      }
      ans += cnt[cur];
      cur = son[cur][1];
    }
  }
}
int GetKth(int k) {
  int cur = rt;
  while (true) {
    if (son[cur][0] && k <= sz[son[cur][0]]) cur = son[
        cur][0];
    else {
      k -= cnt[cur] + sz[son[cur][0]];
      if (k <= 0) return cur;
      cur = son[cur][1];
    }
  }
}
// 获取以r为根节点Splay Tree中的第k大个元素在Splay Tree
    中的位置
int Kth(int r, int k) {
  Pull(r);
  int tmp = sz[son[r][0]] + 1;
  if (tmp == k) return r;
  if (tmp > k) return Kth(son[r][0], k);
  else return Kth(son[r][1], k - tmp);
}
// Insert之后求前驱后继
int GetPrev() {
  int cur = son[rt][0];
  while (son[cur][1]) cur = son[cur][1];
  return cur;
}
int GetNext() {
  int cur = son[rt][1];
  while (son[cur][0]) cur = son[cur][0];
  return cur;
}
// 获取Splay Tree中以o为根节点子树的最小值位置
int GetMin(int o) {
  Pull(o);
  while (son[o][0]) {
    o = son[o][0];
    Pull(o);
  }
  return o;
}
// 获取Splay Tree中以o为根节点子树的最大值位置
int GetMax(int o) {
  Pull(o);
  while (son[o][1]) {
    o = son[o][1];
    Pull(o);
  }
  return o;
}
void Delete(int x) {
  GetRank(x);
  if (cnt[rt] > 1) {
    cnt[rt]--;
    Pull(rt);
    return;
  }
  if (!son[rt][0] && !son[rt][1]) {
    Clear(rt);
    rt = 0;
    return;
  }
  if (!son[rt][0]) {
    int cur = rt;
    rt = son[rt][1];
    fa[rt] = 0;
    Clear(cur);
    return;
  }
  if (!son[rt][1]) {
    int cur = rt;
    rt = son[rt][0];
    fa[rt] = 0;
    Clear(cur);
    return;
  }
  int p = GetPrev(), cur = rt;
  Splay(p);
  fa[son[cur][1]] = p;
  son[p][1] = son[cur][1];
  Clear(cur);
  Pull(rt);
}
/* 维护数组操作 */
// 翻转Splay Tree中l~r区间
void Reverse(int l, int r) {
  int o = Kth(rt, l), Y = Kth(rt, r);
  Splay(o, 0); Splay(Y, o);
  lazy[son[Y][0]] ^= 1;
}
// 建立Splay Tree
void Build(int l, int r, int o) {
  if (l > r) return;
  int m = (l + r) >> 1;
  Build(l, m - 1, m);
  Build(m + 1, r, m);
  fa[m] = o;
```

```
    val[m] = /* 节点权值 */;
    lazy[m] = 0;
    Push(m);
    if (m < o) son[o][0] = m;
    else son[o][1] = m;
  }
  // 输出Splay Tree
  void Print(int o) {
    Pull(o);
    if (son[o][0]) Print(son[o][0]);
    // 哨兵节点判断
    if (val[o] != -inf && val[o] != inf) printf("%d ",
        val[o]);
    if (val[son[o][1]]) Print(son[o][1]);
  }
};
```

3.7 主席树

```
const int maxn = "Edit";
struct FuncSegTree {
  int tot;
  int rt[maxn];
  int lson[maxn * 40], rson[maxn * 40];
  int cnt[maxn * 40];
  int Build(int l, int r) {
    int o = ++tot, m = (l + r) / 2;
    cnt[o] = 0;
    if (l != r) {
      lson[o] = Build(l, m);
      rson[o] = Build(m + 1, r);
    }
    return o;
  }
  int Modify(int prev, int l, int r, int v) {
    int o = ++tot, m = (l + r) / 2;
    lson[o] = lson[prev];
    rson[o] = rson[prev];
    cnt[o] = cnt[prev] + 1;
    if (l != r) {
      if (v <= m) lson[o] = Modify(lson[o], l, m, v);
      else rson[o] = Modify(rson[o], m + 1, r, v);
    }
    return o;
  }
  // 区间[u+1,v]静态第k小
  int Query(int u, int v, int l, int r, int k) {
    if (l == r) return l;
    int m = (l + r) / 2;
    int num = cnt[lson[v]] - cnt[lson[u]];
    if (num >= k) return Query(lson[u], lson[v], l, m, k)
        ;
    else return Query(rson[u], rson[v], m + 1, r, k - num
        );
  }
  // 区间[u+1,v]内[s,t]数量
  int Query(int u, int v, int s, int t, int l, int r) {
    if (s <= l && t >= r) return cnt[v] - cnt[u];
    int m = (l + r) / 2, ret = 0;
    if (s <= m) ret += Query(lson[u], lson[v], s, t, l, m
        );
    if (t > m) ret += Query(rson[u], rson[v], s, t, m +
        1, r);
    return ret;
  }
};
```

3.8 dfs 序

```
const int maxn = "Edit";
std::vector<int> g[maxn];
int in[maxn], out[maxn];
int ele[maxn];
int dfs_clock;
void DfsSeq(int u, int p) {
  in[u] = ++dfs_clock;
  ele[dfs_clock] = u;
  for (int &v : g[u]) {
    if (v == p) continue;
    DfsSeq(v, u);
  }
  out[u] = dfs_clock;
}
```

3.9 ST 表

```
const int maxn = "Edit";
const int maxlog = "Edit";
int n;
int max[maxn][maxlog], min[maxn][maxlog];
void Init(int arr[]) {
  int m = log2(n) + 1;
  for (int i = 1; i <= n; ++i) max[i][0] = min[i][0] =
      arr[i];
  for (int j = 1; j < m; ++j) {
    for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
      max[i][j] = std::max(max[i][j - 1], max[i + (1 << (
          j - 1))][j - 1]);
```

```
      min[i][j] = std::min(min[i][j - 1], min[i + (1 << (
          j - 1))][j - 1]);
    }
  }
}
// 区间[l,r]最大值
int QueryMax(int l, int r) {
  int k = log2(r - l + 1);
  return std::max(max[l][k], max[r - (1 << k) + 1][k]);
}
// 区间[l,r]最小值
int QueryMin(int l, int r) {
  int k = log2(r - l + 1);
  return std::min(min[l][k], min[r - (1 << k) + 1][k]);
}
```

3.10 Link Cut Tree

```
const int maxn = "Edit";
struct LCT {
  int fa[maxn], son[maxn][2];
  int val[maxn], sum[maxn];
  int rev[maxn], stk[maxn];
  void Init(int n) {
    for (int i = 1; i <= n; ++i) scanf("%d", &val[i]);
    for (int i = 1; i <= n; ++i) fa[i] = son[i][0] = son[
        i][1] = rev[i] = 0;
  }
  bool IsRoot(int o) {
    return son[fa[o]][0] != o && son[fa[o]][1] != o;
  }
  bool Get(int o) {
    return son[fa[o]][1] == o;
  }
  // 更新所需维护的信息
  void Pull(int o) {
    sum[o] = val[o] ^ sum[son[o][0]] ^ sum[son[o][1]];
  }
  void Push(int o) {
    if (rev[o] != 0) {
      std::swap(son[o][0], son[o][1]);
      if (son[o][0]) rev[son[o][0]] ^= 1;
      if (son[o][1]) rev[son[o][1]] ^= 1;
      rev[o] ^= 1;
    }
  }
  void Rotate(int o) {
    int p = fa[o], q = fa[p], ck = Get(o);
    if (!IsRoot(p)) son[q][Get(p)] = o;
    fa[o] = q;
    son[p][ck] = son[o][ck ^ 1];
    fa[son[p][ck]] = p;
    son[o][ck ^ 1] = p;
    fa[p] = o;
    Pull(p);
    Pull(o);
  }
  void Splay(int o) {
    int top = 0;
    stk[++top] = o;
    for (int i = o; !IsRoot(i); i = fa[i]) stk[++top] =
        fa[i];
    for (int i = top; i; --i) Push(stk[i]);
    for (int f = fa[o]; !IsRoot(o); Rotate(o), f = fa[o])
      if (!IsRoot(f)) Rotate(Get(o) == Get(f) ? f : o);
  }
  // 将使o成为一条实路径并在同一棵Splay内
  void Access(int o) {
    for (int p = 0; o; p = o, o = fa[o]) {
      Splay(o);
      son[o][1] = p;
      Pull(o);
    }
  }
  // 返回o所在树的根节点编号
  int Find(int o) {
    Access(o);
    Splay(o);
    while (son[o][0]) o = son[o][0];
    return o;
  }
  // 使o成为其所在树的根
  void MakeRoot(int o) {
    Access(o);
    Splay(o);
    rev[o] ^= 1;
  }
  // u,v之间连边,先判不能在同一棵树内
  void Link(int u, int v) {
    MakeRoot(u);
    fa[u] = v;
    Splay(u);
  }
  // 删除u,v之间的边
  void Cut(int u, int v) {
    MakeRoot(u);
    Access(v);
    Splay(v);
    fa[u] = son[v][0] = 0;
  }
  // o节点单点修改
  void Modify(int o, int v) {
    val[o] = v;
    Access(o);
    Splay(o);
```

```
  }
  // u,v路径信息
  int Query(int u, int v) {
    MakeRoot(v);
    Access(u);
    Splay(u);
    return sum[u];
  }
};
```

4 字符串

4.1 马拉车

```
struct Manacher{
    int RL[maxn << 1];
    char s[maxn], t[maxn << 1];
    int getlen(char *s) {
        if (s[strlen(s) - 1] == '\n')   s[strlen(s) - 1]
             = '\0';
        int lens = strlen(s), len = 0;
        t[len++] = '#';
        for (int i = 0; i < lens; ++i) {
            t[len++] = s[i];
            t[len++] = '#';
        }
        int MaxRight = 0, pos = 0, MaxLen = 0;
        for (int i = 0; i < len; ++i) {
            if (i < MaxRight)   RL[i] = min(RL[2 * pos -
                 i], MaxRight - i + 1); // 好多这里写的是
                  MaxRight - i, 个人感觉根据算法思想应该
                 +1计算长度。
            else    RL[i] = 1;
            int l = i - RL[i];
            int r = i + RL[i];
            while (l >= 0 && r < len && t[l] == t[r])   {
                RL[i] += 1;
                l = i - RL[i];
                r = i + RL[i];
            }
            if (RL[i] + i - 1 > MaxRight) {
                MaxRight = RL[i] + i - 1;
                pos = i;
            }
            MaxLen = max(MaxLen, RL[i]);
        }
        return MaxLen - 1;
    }
}manacher;
```

4.2 最小表示法

```
int minRepresent(char *s, int len) {
    int i = 0, j = 1, k = 0;
    while (i < len && j < len && k < len) {
        int t = s[(i+k) % len] - s[(j+k) % len];
        if (t == 0) k++;
        else {
            if (t < 0)  j = max(j+k+1, i+1);
            else i = max(i+k+1, j+1);
```

```
            k = 0;
        }
    }
    return min(i, j);
}
int minRepresent(int start, int end, int len) { // 判断[
     strat, end]是否为最小表示
    int i = 0+start, j = 1+start, k = 0;
    while (i < end && j < end && k < len) {
        int l = i + k;  if (l >= end) l = l - end + start
             ;
        int r = j + k;  if (r >= end) r = r - end + start
             ;
        int t = s[l] - s[r];
        if (t == 0) k++;
        else {
            if (t < 0)  j = max(j+k+1, i+1);
            else i = max(i+k+1, j+1);
            k = 0;
        }
    }
    return min(i, j) == start;
}
```

4.3 扩展 kmp

```
struct exKMP{
    // 字符串下标从0开始
    int nex[maxn], ex[maxn]; //模式串nex, 匹配串ex
    void get_nex(char *str, int len) {
        int i = 0, j, pos;
        nex[0] = len;
        while (str[i] == str[i+1] && i+1 < len) ++i;
        nex[1] = i;
        pos = 1;
        for (int i = 2; i < len; ++i) {
            if (nex[i-pos] + i < nex[pos] + pos) nex[i] =
                  nex[i-pos];
            else {
                j = nex[pos] + pos - i;
                if (j < 0) j = 0;
                while (i+j < len && str[j] == str[j+i])
                     ++j;
                nex[i] = j;
                pos = i;
            }
        }
    }
    void get_ex(char *s1, char *s2) { // s1匹配s2
        int i = 0, j, pos;
        int len1 = strlen(s1);
        int len2 = strlen(s2);
        get_nex(s2, len2);
        while (s1[i] == s2[i] && i < len1 && i < len2) ++
             i;
        ex[0] = i;
        pos = 0;
        for (int i = 1; i < len1; ++i) {
            if (nex[i-pos] + i < ex[pos] + pos) ex[i] =
                 nex[i-pos];
            else {
                j = ex[pos] + pos - i;
                if (j < 0) j = 0;
                while (i+j < len1 && j < len2 && s1[i+j]
                     == s2[j]) ++j;
                ex[i] = j;
                pos = i;
            }
        }
    }
}ek;
```

4.4 字典树

```
struct Trie{
    int nex[maxn][26], cnt[maxn], end[maxn];
    int p, root; // root = 0
    int newnode() {
        memset(nex[p], 0, sizeof(nex[p]));
        cnt[p] = end[p] = 0;
        return p++;
    }
    void init() {
        p = 0;
        root = newnode();
    }
    void add(char *s) {
        int now = root;
        for (int i = 0; s[i]; ++i) {
            if (nex[now][s[i] - 'a'] == 0) nex[now][s[i]
                 - 'a'] = newnode();
            now = nex[now][s[i] - 'a'];
            cnt[now]++;
        }
        end[now] = 1;
    }
    int find(char *s) {
        int now = root;
        for (int i = 0; s[i]; ++i) {
            if (nex[now][s[i] - 'a'] == 0) return 0;
            now = nex[now][s[i] - 'a'];
        }
```

```
        return cnt[now];
    }
}trie;
```

4.5 回文树

```
struct Palindrome_Tree{
    int nex[maxn][26];
    int fail[maxn], cnt[maxn], num[maxn]; // num 记录每个
        节点右端点的表示回文串的个数
    int len[maxn], S[maxn];                  // cnt 记录每
        个节点表示的回文串出现的次数
    int last, n, p;
    int newnode(int l) { // 新建节点
        for (int i = 0; i < 26; ++i) nex[p][i] = 0;
        cnt[p] = num[p] = 0;
        len[p] = l;
        return p++;
    }
    void init() { // 初始化
        p = 0;
        newnode(0), newnode(-1); // 新建奇根和偶根
        last = n = 0;
        S[n] = -1;
        fail[0] = 1; // 偶根指向
    }
    int get_fail(int x) { // 求fail
        while (S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }
    void add(int c) { // 添加节点
        c -= 'a';
        S[++n] = c;
        int cur = get_fail(last);
        if (!nex[cur][c]) {
            int now = newnode(len[cur] + 2);
            fail[now] = nex[get_fail(fail[cur])][c];
            nex[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        last = nex[cur][c];
        cnt[last]++;
    }
    void build(char *buf, int lens) {
        init();
        for (int i = 0; i < lens; ++i) add(buf[i]);
    }
    void count() { // 求cnt
        for (int i = p - 1; i >= 0; --i) cnt[fail[i]] +=
            cnt[i];
    }
}Tree;
```

4.6 哈希

```
struct Hash{
    // mod 402653189, 805306457, 1610612741, 1e9+7
    // base 131, 233
    long long p[maxn], hash[maxn], base = 131;
    long long getHash(int l, int r) {
        long long ans = (hash[r] - hash[l-1] * p[r-l+1])
            % mod;
        return (ans + mod) % mod;
    }
    void init(string s) {
        int n = s.size();
        p[0] = 1;
        for (int i = 1; i <= n; ++i) p[i] = p[i - 1] *
            base % mod;
        for (int i = 1; i <= n; ++i) {
            hash[i] = (hash[i - 1] * base % mod + (s[i-1]
                - 'a' + 1)) % mod;
        }
    }
}hash;
```

4.7 后缀自动机 (SAM)

```
struct SAM{
    int trans[maxn<<1][26], slink[maxn<<1], maxlen[maxn
        <<1];
    // 用来求endpos
    int indegree[maxn<<1], endpos[maxn<<1], rank[maxn
        <<1], ans[maxn<<1];
    // 计算所有子串的和(0-9表示)
    long sum[maxn<<1];
    int last, now, root, len;
    inline void newnode (int v) {
        maxlen[++now] = v;
    }
    inline void extend(int c) {
        newnode(maxlen[last] + 1);
        int p = last, np = now;
        // 更新trans
        while (p && !trans[p][c]) {
            trans[p][c] = np;
            p = slink[p];
        }
        if (!p) slink[np] = root;
        else {
```

```
            int q = trans[p][c];
            if (maxlen[p] + 1 != maxlen[q]) {
                // 将q点拆出nq, 使得maxlen[p] + 1 ==
                     maxlen[q]
                newnode(maxlen[p] + 1);
                int nq = now;
                memcpy(trans[nq], trans[q], sizeof(trans[
                     q]));
                slink[nq] = slink[q];
                slink[q] = slink[np] = nq;
                while (p && trans[p][c] == q) {
                    trans[p][c] = nq;
                    p = slink[p];
                }
            }else slink[np] = q;
        }
        last = np;
        // 初始状态为可接受状态
        endpos[np] = 1;
    }
    inline void build(char *s) {
        // scanf("%s", s);
        len = strlen(s);
        root = last = now = 1;
        for (int i = 0; i < len; ++i) extend(s[i] - '0');
              // extend(s[i] - '1');
    }
    // 计算所有子串的和 (0-9表示)
    inline long getSum() {
        // 拓扑排序
        for (int i = 1; i <= now; ++i) indegree[ maxlen[i
             ] ]++;
        for (int i = 1; i <= now; ++i) indegree[i] +=
             indegree[i-1];
        for (int i = 1; i <= now; ++i) rank[ indegree[
             maxlen[i] ]-- ] = i;
        mem(endpos, 0);
        endpos[1] = 1; // 从根节点向后求有效的入度
        for (int i = 1; i <= now; ++i) {
            int x = rank[i];
            for (int j = 0; j < 10; ++j) {
                int nex = trans[x][j];
                if (!nex) continue;
                endpos[nex] += endpos[x]; // 有效入度
                long num = (sum[x] * 10 + endpos[x] * j)
                     % mod;
                sum[nex] = (sum[nex] + num) % mod; // 状
                     态转移
            }
        }
        long long ans = 0;
        for (int i = 2; i <= now; ++i) ans = (ans + sum[i
             ]) % mod;
        return ans;
    }
    inline void getEndpos() {
        // topsort
        for (int i = 1; i <= now; ++i) indegree[ maxlen[i
             ] ]++; // 统计相同度数的节点的个数
        for (int i = 1; i <= now; ++i) indegree[i] +=
             indegree[i-1]; // 统计度数小于等于 i 的节点
             的总数
        for (int i = 1; i <= now; ++i) rank[ indegree[
             maxlen[i] ]-- ] = i; // 为每个节点编号, 节
             点度数越大编号越靠后
        // 从下往上按照slik更新
        for (int i = now; i >= 1; --i) {
            int x = rank[i];
            endpos[slink[x]] += endpos[x];
        }
    }
    // 求不同的子串种类
    inline long long all () {
        long long ans = 0;
        for (int i = root+1; i <= now; ++i) {
            ans += maxlen[i] - maxlen[ slink[i] ];
        }
        return ans;
    }
    // 长度为K的字符串有多种, 求出现次数最多的次数
    inline void get_Maxk() {
        getEndpos();
        for (int i = 1; i <= now; ++i) {
            ans[maxlen[i]] = max(ans[maxlen[i]], endpos[i
                 ]);
        }
        for (int i = len; i >= 1; --i) ans[i] = max(ans[i
             ], ans[i+1]);
        for (int i = 1; i <= len; ++i) //cout << ans[i]
             << endl;
            printf("%d\n", ans[i]);
    }
}sam;
```

4.8 后缀数组

```
struct SuffixArray{ // 下标1
    int cntA[maxn], cntB[maxn], A[maxn], B[maxn];
    int Sa[maxn], tsa[maxn], height[maxn], Rank[maxn]; //
         Sa[i] 排名第i的下标, Rank[i] 下标i的排名
    int n, dp[maxn][21];
    void init(char *buf, int len) { // 预处理, sa, rank,
         height
        n = len;
        for (int i = 0; i < 500; ++i) cntA[i] = 0;
```

```
    for (int i = 1; i <= n; ++i) cntA[(int)buf[i]]++;
    for (int i = 1; i < 500; ++i) cntA[i] += cntA[i
        -1];
    for (int i = n; i >= 1; --i) Sa[ cntA[(int)buf[i
        ]]-- ] = i;
    Rank[ Sa[1] ] = 1;
    for (int i = 2; i <= n; ++i) {
        Rank[Sa[i]] = Rank[Sa[i-1]];
        if (buf[Sa[i]] != buf[Sa[i-1]]) Rank[Sa[i
            ]]++;
    }
    for (int l = 1; Rank[Sa[n]] < n; l <<= 1) {
        for (int i = 0; i <= n; ++i) cntA[i] = 0;
        for (int i = 0; i <= n; ++i) cntB[i] = 0;
        for (int i = 1; i <= n; ++i) {
            cntA[ A[i] = Rank[i] ]++;
            cntB[ B[i] = (i + l <= n) ? Rank[i+l] :
                0]++;
        }
        for (int i = 1; i <= n; ++i) cntB[i] += cntB[
            i-1];
        for (int i = n; i >= 1; --i) tsa[ cntB[B[i
            ]]-- ] = i;
        for (int i = 1; i <= n; ++i) cntA[i] += cntA[
            i-1];
        for (int i = n; i >= 1; --i) Sa[ cntA[A[tsa[i
            ]]]-- ] = tsa[i];
        Rank[ Sa[1] ] = 1;
        for (int i = 2; i <= n; ++i) {
            Rank[Sa[i]] = Rank[Sa[i-1]];
            if (A[Sa[i]] != A[Sa[i-1]] || B[Sa[i]] !=
                B[Sa[i-1]]) Rank[Sa[i]]++;
        }
    }
    for (int i = 1, j = 0; i <= n; ++i) {
        if (j) --j;
        int tmp = Sa[Rank[i] - 1];
        while (i + j <= n && tmp + j <= n && buf[i+j]
            == buf[tmp+j]) ++j;
        height[Rank[i]] = j;
    }
}
void st() {
    for (int i = 1; i <= n; ++i) {
        dp[i][0] = height[i];
    }
    for (int j = 1; j <= log2(n); ++j) {
        for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 <<
                (j - 1))][j - 1]);
        }
    }
}
int rmq(int l, int r) {
    int len = r - l + 1;
    int x = log2(len);
    return min(dp[l][x], dp[r - (1 << x) + 1][x]);
}
int lcp(int x, int y) { // 最长公共前缀
    int l = Rank[x];
    int r = Rank[y];
    if (l > r) swap(l, r);
    return rmq(l+1, r);
}
int getnum() { // 字串的个数
    int ans = 0;
    for (int i = 1; i <= n; ++i) {
        ans += n - Sa[i] + 1 - height[i];
    }
    return ans;
}
}S;
```

4.9 kmp

```
struct KMP{ // 下标0
    int nex[maxn];
    void get_nex(char *buf, int len) {
        nex[0] = -1;
        int i = 0, j = -1;
        while (i < len) {
            if (j == -1 || buf[i] == buf[j]) nex[++i] =
                ++j;
            else j = nex[j];
        }
    }
    int get_kmp(char *buf1, char *buf2) { // buf1匹配串,
        buf2模式串
        int len1 = strlen(buf1), len2 = strlen(buf2);
        get_nex(buf2, len2);
        int cnt = 0, i = 0, j = 0;
        while (i < len1) {
            if (j == -1 || buf1[i] == buf2[i]) ++i, ++j;
            else j = nex[j];
            if (j == len2) cnt++, j = nex[j];
        }
        return cnt; // 匹配个数
    }
    // (len - nex[len]) 最小循环节, 前提len % (len - nex[
        len]) = 0
}kmp;
```

4.10 AC 自动机

```
struct Trie{
    int nex[maxn][26], fail[maxn], end[maxn];
    int root, p;
    inline int newnode() {
        for (int i = 0; i < 26; ++i) {
            nex[p][i] = -1;
        }
        end[p++] = 0;
        return p - 1;
    }
    inline void init() {
        p = 0;
        root = newnode();
    }
    inline void insert(char *buf) {
        int now = root;
        for (int i = 0; buf[i]; ++i) {
            if (nex[now][buf[i]-'a'] == -1)
                nex[now][buf[i]-'a'] = newnode();
            now = nex[now][buf[i]-'a'];
        }
        end[now]++;
    }
    inline void build() {
        queue<int> que;
        fail[root] = root;
        for (int i = 0; i < 26; ++i) {
            if (nex[root][i] == -1)
                nex[root][i] = root;
            else {
                fail[nex[root][i]] = root;
                que.push(nex[root][i]);
            }
        }
        while (!que.empty()) {
            int now = que.front();
            que.pop();
            for (int i = 0; i < 26; ++i) {
                if (nex[now][i] == -1)
                    nex[now][i] = nex[fail[now]][i];
                else {
                    fail[nex[now][i]] = nex[fail[now]][i
                        ];
                    que.push(nex[now][i]);
                }
            }
        }
    }
    long long num[maxn], dp[maxn]; // num记录节点i匹配的
        个数, dp辅助得到所有适配数量
    long long dfs(int now) {
        if (now == root) return 0;
        if (dp[now] != -1) return dp[now];
        return dp[now] = end[now] + dfs(fail[now]);
    }
    inline void solve(char *buf) {
        fill(num, num+maxn, 0);
        fill(dp, dp+maxn, -1);
        int now = root;
        for (int i = 0; buf[i]; ++i) {
            now = nex[now][buf[i]-'a'];
            num[i] = dfs(now);
        }
    }
    inline long long query(char *buf) {
        int now = root;
        long long cnt = 0;
        for (int i = 0; buf[i]; ++i) {
            now = nex[now][buf[i]-'a'];
            int tmp = now;
            while (tmp != root && end[tmp] != -1) {
                cnt += end[tmp];
                end[tmp] = -1; // 统计种类, 加速
                tmp = fail[tmp];
            }
        }
        return cnt;
    }
}L, R;
```

5 图论

5.1 费用流

```
// SPFA
int path[maxn], dis[maxn], head[maxn], vis[maxn], cnt;
void init() {
    cnt = 0;
    memset(head, -1, sizeof(head));
}
struct ac{
    int v, c, cost, nex;
}edge[maxn << 10]; // 根据题目要求计算

void addedge(int u, int v, int c, int cost) {
    // 正向建边
    edge[cnt] = {v, c, cost, head[u]};
    head[u] = cnt++;
    // 反向建边
    edge[cnt] = {u, 0, -cost, head[v]};
    head[v] = cnt++;
}
int spfa(int s, int e) {
    memset(vis, 0, sizeof(vis));
    memset(dis, inf, sizeof(dis)); // 记录从s点出发到每个点的费用和最小值
    memset(path, -1, sizeof(path)); // 记录更新当前点的边在edge中的下标
    queue<int> que;
    que.push(s);
    dis[s] = 0;
    vis[s] = 1;
    while (!que.empty()) {
        int u = que.front();
        que.pop();
        vis[u] = 0;
        // 遍历u的所有出边
        for (int i = head[u]; i != -1; i = edge[i].nex) {
            int v = edge[i].v;
            int c = edge[i].c;
            int cost = edge[i].cost;
            // 判断是否更新v点
            if (dis[v] > dis[u] + cost && c > 0) {
                dis[v] = dis[u] + cost; // 更新最小费用
                path[v] = i;
                if (vis[v]) continue;
                vis[v] = 1;
                que.push(v);
            }
        }
    }
    return dis[e] != inf; // 判断s能否到达e
}
int MincostMaxflow(int s, int e, int &cost) {
    int maxflow = 0;
    while (spfa(s, e)) { // 搜先spfa看是否存在增广路，如果存在求一条费用和最小的一条
        int flow = inf;
        // 遍历增广路上的边，取最小的流量flow
        // path存的是那条边更新到这个点，i = 这个点在edge中的下标
        // edge[i^1].v 通过反向边得到前驱节点
        for (int i = path[e]; i != -1; i = path[edge[i^1].v]) {
            flow = min(flow, edge[i].c); // 取最小的流量
        }
        // 得到最小流量flow之后，更改正反向的流量
        for (int i = path[e]; i != -1; i = path[edge[i^1].v]) {
            edge[i].c -= flow;
            edge[i^1].c += flow;
            cost += flow * edge[i].cost;
        }
        maxflow += flow;
    }
    return maxflow; // 返回最大流
}
// Dijkstra + 链式
int preE[maxn], preV[maxn], dis[maxn], head[maxn], vis[maxn], h[maxn], cnt;
void init() {
    cnt = 0;
    memset(head, -1, sizeof(head));
}
struct ac{
    int v, c, cost, nex;
}edge[maxn << 8];

void addedge(int u, int v, int c, int cost) {
    edge[cnt] = {v, c, cost, head[u]};
    head[u] = cnt++;
    edge[cnt] = {u, 0, -cost, head[v]};
    head[v] = cnt++;
}
int Dijkstra(int s, int e) {
    memset(dis, inf, sizeof(dis));
    preE[s] = -1, dis[s] = 0;
    priority_queue< pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>> >que;
    que.push(pair<int,int>(0, s));
    while (!que.empty()) {
        pair<int, int> top = que.top();
        que.pop();
        int u = top.second;
        if (dis[u] < top.first) continue;
        for (int i = head[u]; i != -1; i = edge[i].nex) {
```

```
            int v = edge[i].v;
            int cost = edge[i].cost;
            int c = edge[i].c;
            if (c > 0 && dis[v] > dis[u] + cost + h[u] -
                 h[v]) {
                dis[v] = dis[u] + cost + h[u] - h[v];
                preE[v] = i;
                preV[v] = u;
                que.push(pair<int,int>(dis[v], v));
            }
        }
    }
    return dis[e] != inf;
}
int MincostMaxflow(int s, int e, int &cost) {
    int maxflow = 0;
    memset(h, 0, sizeof(h));
    while (Dijkstra(s, e)) { // 搜先spfa看是否存在增广
         路, 如果存在求一条费用和最小的一条
        for (int i = 0; i <= e; ++i) h[i] += dis[i];
        int flow = inf;
        for (int i = e; i != s; i = preV[i]) {
            flow = min(flow, edge[preE[i]].c); // 取最小
                 的流量
        }
        for (int i = e; i != s; i = preV[i]) {
            edge[preE[i]].c -= flow;
            edge[preE[i]^1].c += flow;
        }
        cost += flow * h[e];
        maxflow += flow;
    }
    return maxflow; // 返回最大流
}
// Dijkstra + vector
int preE[maxn], preV[maxn], dis[maxn], h[maxn];
struct ac{
    int v, c, cost, nex;
};
vector<ac> g[maxn];
void init() {
    for (int i = 0; i < maxn; ++i) g[i].clear();
}
void addedge(int u, int v, int c, int cost) {
    g[u].push_back({v, c, cost, (int)g[v].size()});
    g[v].push_back({u, 0, -cost, (int)g[u].size()-1});
}
int Dijkstra(int s, int e) {
    priority_queue< pair<int,int>, vector<pair<int,int>>,
          greater<pair<int,int>> >que;
    que.push(pair<int,int>(0, s));
    memset(dis, inf, sizeof(dis));
    dis[s] = 0;
    while (!que.empty()) {
        pair<int, int> top = que.top();
        que.pop();
        int u = top.second;
        if (dis[u] < top.first) continue;
        for (int i = 0; i < (int)g[u].size(); ++i) {
            int v = g[u][i].v;
            int cost = g[u][i].cost;
            int c = g[u][i].c;
            if (c > 0 && dis[v] > dis[u] + cost + h[u] -
                 h[v]) {
                dis[v] = dis[u] + cost + h[u] - h[v];
                preE[v] = i;
                preV[v] = u;
                que.push(pair<int,int>(dis[v], v));
            }
        }
    }
    return dis[e] != inf;
}
```

5.2 网络流

```
struct ac{
    int v, c, nex;
}edge[maxn << 10]; // 根据题目要求计算
int s, e;
int head[maxn], dis[maxn], curedge[maxn], cnt;
void init() {
    cnt = 0;
    memset(head, -1, sizeof(head));
}
void addedge(int u, int v, int c) {
    // 正向建边
    edge[cnt] = {v, c, head[u]};
    head[u] = cnt++;
    // 反向建边, 流量为0
    edge[cnt] = {u, 0, head[v]};
    head[v] = cnt++;
}
bool bfs() {
    queue<int> que;
    que.push(s);
    memset(dis, 0, sizeof(dis)); // 对图进行分层
    dis[s] = 1;
    while (!que.empty()) {
        int u = que.front();
        que.pop();
        for (int i = head[u]; i != -1; i = edge[i].nex) {
            int v = edge[i].v;
            int c = edge[i].c;
```

```
            // 如果节点v已经分过层或者u->v流量为0,
                continue
            if (dis[v] || c == 0) continue;
            dis[v] = dis[u] + 1; // 对v进行标记并加入队列
            que.push(v);
        }
    }
    return dis[e] > 0;  // 判断是否存在增广路, s是否能到
        达e
}
int dfs(int u, int flow) { // 增广路走到u点的最小流量为
    flow
    if (u == e || flow == 0) return flow;
    // 遍历u的所有出边
    for (int &i = curedge[u]; i != -1; i = edge[i].nex) {
        // 当前弧优化
        int v = edge[i].v;
        int c = edge[i].c;
        // 判断能否u->v增广
        if (dis[v] != dis[u] + 1 || c == 0) continue;
        int d = dfs(v, min(flow, c));
        if (d > 0) { // 找到一条增广路, 修改增广路上的正
            反向边
            edge[i].c -= d;
            edge[i^1].c += d;
            return d;
        }
    }
    dis[u] = -1; // // 炸点优化
    return 0;
}
int Dinic() {
    int sum = 0, d;
    while (bfs()) { // 判读是否存在增广路
        for (int i = 0; i <= e; ++i) curedge[i] = head[i
            ]; // copy head数组, 在dfs中可以直接得到下一
            条没有被增广过的边
        while ((d = dfs(s, inf)) > 0) sum += d; // 多次
            dfs找增广路
    }
    return sum;
}
```

5.3 次小生成树

```
// Kruskal
int n, m;
struct ac{
    int u, v, w, flag;
    bool operator <(ac t) {
        return w < t.w;
    }
}g[maxn*maxn];
vector<int> son[maxn];
int pre[maxn], dis[maxn][maxn];
int find (int x) {
    return (pre[x] == x) ? x : pre[x] = find(pre[x]);
}
void Kruskal() {
    for (int i = 0; i <= n; ++i) {
        son[i].clear();
        son[i].push_back(i);
        pre[i] = i;
    }
    sort(g, g+m);
    int sum = 0;
    int cnt = 0;
    for (int i = 0; i < m; ++i) {
        if (cnt == n+1) break;
        int fx = find(g[i].u);
        int fy = find(g[i].v);
        if (fx == fy) continue;
        g[i].flag = 1;
        sum += g[i].w;
        cnt++;
        int lenx = son[fx].size();
        int leny = son[fy].size();
        if (lenx < leny) {
            swap(lenx, leny);
            swap(fx, fy);
        }
        // 更新两点的距离最大值
        for (int j = 0; j < lenx; ++j) {
            for (int k = 0; k < leny; ++k) {
                dis[son[fx][j]][son[fy][k]] = dis[son[fy
                    ][k]][son[fx][j]] = g[i].w;
            }
        }
        pre[fy] = fx;
        //合并子树
        for (int j = 0; j < leny; ++j) {
            son[fx].push_back(son[fy][j]);
        }
        son[fy].clear();
    }
    int ans = inf;
    for (int i = 0; i < m; ++i) {
        if (g[i].flag) continue;
        ans = min(ans, sum + g[i].w - dis[g[i].u][g[i].v
            ]);
    }
    printf("%d %d\n", sum, ans);
}
// Prim
int n, m;
```

```
int g[maxn][maxn], val[maxn], vis[maxn], dis[maxn];
int pre[maxn], maxd[maxn][maxn];
bool used[maxn][maxn];
void prim(int s) {
    mem(maxd, 0);
    mem(vis, 0);
    mem(used, 0);
    for (int i = 1; i <= n; ++i) {
        dis[i] = g[s][i];
        pre[i] = s;
    }
    vis[s] = 1;
    int sum = 0, cnt = 0;
    for (int i = 1; i < n; ++i) {
        int u = -1, MIN = inf;
        for (int j = 1; j <= n; ++j) {
            if (vis[j]) continue;
            if (MIN > dis[j]) {
                MIN = dis[j];
                u = j;
            }
        }
        if (u == -1) break;
        vis[u] = 1;
        sum += MIN;
        cnt++;
        used[pre[u]][u] = used[u][pre[u]] = 1;
        maxd[u][pre[u]] = maxd[pre[u]][u] = MIN;
        for (int j = 1; j <= n; ++j) {
            if (j == u) continue;
            if (vis[j]) {
                maxd[u][j] = maxd[j][u] = max(maxd[pre[u
                    ]][j], MIN);
            }
            if (vis[j] == 0 && dis[j] > g[u][j]) {
                dis[j] = g[u][j];
                pre[j] = u;
            }
        }
    }
    if (cnt != n-1) {
        puts("No way");
    }
    int ans = inf;
    for (int i = 1; i <= n; ++i) {
        for (int j = i+1; j <= n; ++j) {
            if (used[i][j]) continue;
            ans = min(ans, sum + g[i][j] - maxd[i][j]);
        }
    }
    printf("%d %d\n", sum, ans);
}
```

5.4 最小树形图

```
struct ac{
    int u, v, w;
};
vector<ac> g(maxn);
int pre[maxn], vis[maxn], id[maxn], in[maxn];
int zhuliu(int rt, int n, int m) {
    int ans = 0, u, v, w;
    while (1) {
        for (int i = 0; i < n; ++i) in[i] = inf;
        for (int i = 0; i < m; ++i) {
            u = g[i].u; v = g[i].v; w = g[i].w;
            if (u != v && w < in[v]) {
                pre[v] = u;
                in[v] = w;
                // if (u == rt) pos = i; // 记录前驱，输
                    出序号最小的根
            }
        }
        for (int i = 0; i < n; ++i) {
            if (i != rt && in[i] == inf) return -1;
        }
        int cnt = 0;
        mem(id, -1);
        mem(vis, -1);
        in[rt] = 0;
        for (int i = 0; i < n; ++i) {
            ans += in[i];
            u = i;
            while (vis[u] != i && id[u] == -1 && u != rt)
                {
                vis[u] = i;
                u = pre[u];
            }
            if (u != rt &&id[u] == -1) {
                v = pre[u];
                while (v != u) {
                    id[v] = cnt;
                    v = pre[v];
                }
                id[u] = cnt++;
            }
        }
        if (cnt == 0) break;
        for (int i = 0; i < n; ++i) {
            if (id[i] == -1) id[i] = cnt++;
        }
        for (int i = 0; i < m; ++i) {
            v = g[i].v;
            g[i].u = id[g[i].u];
            g[i].v = id[g[i].v];
            if (g[i].u != g[i].v) g[i].w -= in[v];
```

```
        }
        n = cnt;
        rt = id[rt];
    }
    return ans;
}
```

5.5 拓扑排序

```
vector<int> g[maxn];
struct Topsort{ // 下标1
    priority_queue<int, vector<int>, greater<int>> q1; //
        字典序小，正向建图
    priority_queue<int, vector<int>> q2; // 编号小的优先
        级高，反向建图
    int in[maxn], order[maxn], n, cnt;
    void init () {
        fill(in, in+n+1, 0);
        for (int i = 1; i <= n; ++i) {
            for (int j = 0; j < (int)g[i].size(); ++j) {
                in[g[i][j]]++;
            }
        }
    }
    int min_lex (int len) {
        n = len;
        cnt = 0;
        init();
        for (int i = 1; i <= n; ++i)
            if (in[i] == 0) q1.push(i);
        while (!q1.empty()) {
            int u = q1.top();
            q1.pop();
            order[++cnt] = u;
            for (int j = 0; j < (int)g[u].size(); ++j) {
                int v = g[u][j];
                in[v]--;
                if (in[v] == 0) q1.push(v);
            }
        }
        return cnt == n;
    }
    int min_num (int len) {
        cnt = n = len;
        init();
        for (int i = 1; i <= n; ++i)
            if (in[i] == 0) q2.push(i);
        while (!q2.empty()) {
            int u = q2.top();
            q2.pop();
            order[cnt--] = u;
            for (int j = 0; j < (int)g[u].size(); ++j) {
                int v = g[u][j];
                if (--in[v] == 0) q2.push(v);
            }
        }
        return cnt == 0;
    }
}topsort;
```

5.6 Tarjan

```
// 强联通分量
int dfn[maxn], low[maxn], Stack[maxn], inStack[maxn],
    belong[maxn], in[maxn], ts, cnt, len;
void init(int n) {
    for (int i = 1; i <= n; ++i) g[i].clear();
    ts = cnt = len = 0;
    fill(dfn, dfn+n+1, 0);
    fill(inStack, inStack+n+1, 0);
}
void tarjan(int u) {
    dfn[u] = low[u] = ++ts;
    inStack[u] = 1;
    Stack[len++] = u;
    for (int i = 0; i < (int)g[u].size(); ++i) {
        int v = g[u][i];
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }else if (inStack[v]) low[u] = min(low[u], dfn[v
            ]);
    }
    if (dfn[u] == low[u]) {
        cnt++;
        while (1) {
            int top = Stack[--len];
            belong[top] = cnt;
            inStack[top] = 0;
            if (top == u) break;
        }
    }
}
for (int i = 1; i <= n; ++i) {
    if (dfn[i]) continue;
    tarjan(i);
}

// 双连通分量
vector<int> g[maxn];
int dfn[maxn], low[maxn], Stack[maxn], inStack[maxn];
int len, cnt, ts;
```

```
void init(int n) {
    len = cnt = ts = 0;
    for (int i = 1; i <= n; ++i) g[i].clear();
    fill(dfn, dfn+n+1, 0);
}
void tarjan(int u, int fa) {
    dfn[u] = low[u] = ++ts;
    Stack[len++] = u;
    for (int i = 0; i < (int)g[u].size(); ++i) {
        int v = g[u][i];
        if (v == fa) continue;
        if (!dfn[v]) {
            // Stack[len++] = {u, v};
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (dfn[u] <= low[v]) {
                fill(inStack, inStack+n+1, 0);
                inStack[u] = 1;
                while (1) {
                    int top = Stack[--len];
                    inStack[top] = 1; // 记录每次的连通分
                        量中的点
                    if (top == v) break; // top.u == u &&
                         top.v == top.v
                }
                // other check()
            }
        }else low[u] = min(low[u], dfn[v]);
    }
}
```

5.7 Kruskal 重构树

```
struct ac{
    int u, v, w;
    bool operator < (const ac &t) {
        return w < t.w;
    }
}edge[maxn];
struct reset_kruskal{
    struct ac{
        int v, nex;
    }edge[maxn];
    int head[maxn], pre[maxn], cnt, n;
    int dep[maxn], vis[maxn], fa[maxn][31], weight[maxn];
    void init(int t) {
        n = t;
        cnt = 0;
        for (int i = 0; i <= n; ++i) pre[i] = i;
        fill(head, head+n+1, -1);
        fill(vis, vis+n+1, 0);
    }
    void add(int u, int v) {
        edge[cnt] = {v, head[u]};
        head[u] = cnt++;
    }
    void dfs (int u) { // 预处理lca
        vis[u] = 1;
        for (int i = 1; i <= log2(n); ++i) {
            if (fa[u][i-1] == 0) break;
            fa[u][i] = fa[fa[u][i-1]][i-1];
        }
        for (int i = head[u]; ~i; i = edge[i].nex) {
            int v = edge[i].v;
            dep[v] = dep[u] + 1;
            fa[v][0] = u;
            dfs(v);
        }
    }
    int lca(int u, int v) {
        if (dep[u] < dep[v]) swap(u, v);
        int det = dep[u] - dep[v];
        for (int i = 0; i <= log2(det); ++i) {
            if (det & (1 << i)) u = fa[u][i];
        }
        if (u == v) return u;
        for (int i = log2(dep[u]); i >= 0; --i) {
            if (fa[u][i] != fa[v][i]) {
                u = fa[u][i];
                v = fa[v][i];
            }
        }
        return fa[u][0];
    }
    int find (int x) {
        int t = x;
        while (x != pre[x]) x = pre[x];
        while (t != pre[t]) {
            int fa = pre[t];
            pre[t] = x;
            t = fa;
        }
        return x;
    }
}kru;
```

5.8 Dinic

```
struct ac{
    int v, c, pre;
}edge[maxn<<6];
int s, e;
```

```
int head[maxn<<1], dis[maxn<<1], curedge[maxn<<1], cnt;
void init() {
    mem(head, -1);
    cnt = 0;
}
void addedge(int u, int v, int c) { // 记得双向边
    edge[cnt] = {v, c, head[u]};
    head[u] = cnt++;
}
bool bfs() {
    queue<int> que;
    que.push(s);
    mem(dis, 0);
    dis[s] = 1;
    while (!que.empty()) {
        int f = que.front();
        que.pop();
        for (int i = head[f]; i != -1; i = edge[i].pre) {
            if (dis[edge[i].v] || edge[i].c == 0)
                continue;
            dis[edge[i].v] = dis[f] + 1;
            que.push(edge[i].v);
        }
    }
    return dis[e] > 0;
}
int dfs(int now, int flow) {
    if (now == e || flow == 0) return flow;
    for (int &i = curedge[now]; i != -1; i = edge[i].pre)
          { // 当前弧优化
        if (dis[edge[i].v] != dis[now] + 1 || edge[i].c
             == 0) continue;
        int d = dfs(edge[i].v, min(flow, edge[i].c));
        if (d > 0) {
            edge[i].c -= d;
            edge[i^1].c += d;
            return d;
        }
    }
    dis[now] = -1; // // 炸点优化
    return 0;
}
int Dinic() {
    int sum = 0, d;
    while (bfs()) {
        for (int i = 0; i <= e; ++i) curedge[i] = head[i
             ];
        while (d = dfs(s, inf)) sum += d;
    }
    return sum;
}
```

6 其它

6.1 闰年

```
bool IsLeepYear(int y) {
  return (!(y % 4) && (y % 100)) || !(y % 400);
}
```

6.2 蔡勒公式

```
// 返回y年m月d日是星期几
int Zeller(int y, int m, int d) {
  if (m == 1 || m == 2) {
    --y;
    m += 12;
  }
  int c = y / 100;
  y %= 100;
  //1582年10月4日之前
  return ((y + y / 4 + c / 4 - 2 * c + 13 * (m + 1) / 5 +
       d + 2) % 7) + 7 % 7;
  //1582年10月4日之后
  return ((y + y / 4 + c / 4 - 2 * c + 26 * (m + 1) / 10
       + d - 1) % 7 + 7) % 7;
}
```

6.3 莫队算法

6.3.1 静态莫队

```
const int maxn = "Edit";
// 静态莫队算法求区间不同数字数量
struct MoCap {
  int n, m;
  int block;
  int arr[maxn];
  struct query { int l, r, id; };
  query q[maxn];
  int cnt[maxn << 1];
  int cur;
  int ans[maxn];
  void Add(int x) {
    cur += (++cnt[arr[x]] == 1);
  }
  void Del(int x) {
    cur -= (--cnt[arr[x]] == 0);
  }
```

```
  void Solve() {
    scanf("%d%d", &n, &m);
    block = std::sqrt(n);
    for (int i = 1; i <= n; ++i) scanf("%d%d", &arr[i]);
    for (int i = 1; i <= m; ++i) {
      scanf("%d%d", &q[i].l, &q[i].r);
      q[i].id = i;
    }
    std::sort(q + 1, q + m + 1, [&](query k1, query k2) {
          return (k1.l / block) == (k2.l / block) ? k1.r
        < k2.r : k1.l < k2.l; });
    int l = 0, r = 0;
    for (int i = 1; i <= m; ++i) {
      while (l < q[i].l) Del(l++);
      while (l > q[i].l) Add(--l);
      while (r < q[i].r) Add(++r);
      while (r > q[i].r) Del(r--);
      ans[q[i].id] = cur;
    }
    for (int i = 1; i <= m; ++i) printf("%d\n", ans[i]);
  }
}mo;
```

6.3.2 带修莫队

```
const int maxn = "Edit";
// 动态莫队算法求区间不同数字数量（支持单点修改）
struct MoCap {
  int n, m;
  int block;
  int arr[maxn];
  struct query { int l, r, pre, id; };
  int q_tot;
  query q[maxn];
  struct change { int pos, val; };
  int c_tot;
  change c[maxn];
  int cnt[maxn << 7];
  int cur;
  int ans[maxn];
  void Add(int x) {
    cur += (++cnt[arr[x]] == 1);
  }
  void Del(int x) {
    cur -= (--cnt[arr[x]] == 0);
  }
  void Modify(int x, int i) {
    if (c[x].pos >= q[i].l && c[x].pos <= q[i].r) {
      cur -= (--cnt[arr[c[x].pos]] == 0);
      cur += (++cnt[c[x].val] == 1);
    }
    std::swap(c[x].val, arr[c[x].pos]);
  }
  void Solve() {
    scanf("%d%d", &n, &m);
    block = std::sqrt(n);
    for (int i = 1; i <= n; ++i) scanf("%d", &arr[i]);
    for (int i = 1; i <= m; ++i) {
      char op; getchar();
      scanf("%c", &op);
      if (op == 'Q') {
        int l, r; scanf("%d%d", &l, &r);
        q[++q_tot] = (query){l, r, c_tot, q_tot};
      }
      else {
        int p, v; scanf("%d%d", &p, &v);
        c[++c_tot] = (change){p, v};
      }
    }
    std::sort(q + 1, q + q_tot + 1, [&](query k1, query
         k2) {
      if ((k1.l / block) == k2.l / block) {
        if ((k1.r / block) == (k2.r / block)) return k1.
            pre < k2.pre;
        return k1.r < k2.r;
      }
      return k1.l < k2.l;
    });
    int l = 1, r = 0, t = 0;
    for (int i = 1; i <= q_tot; ++i) {
      while (l < q[i].l) Del(l++);
      while (l > q[i].l) Add(--l);
      while (r < q[i].r) Add(++r);
      while (r > q[i].r) Del(r--);
      while (t < q[i].pre) Modify(++t, i);
      while (t > q[i].pre) Modify(t--, i);
      ans[q[i].id] = cur;
    }
    for (int i = 1; i <= q_tot; ++i) printf("%d\n", ans[i
        ]);
  }
}mo;
```

6.4 快读

```
// 普通快读
template <typename t>
inline bool Read(t &ret) {
  char c; int sgn;
  if (c = getchar(), c == EOF) return false;
  while (c != '-' && (c < '0' || c > '9')) c = getchar();
  sgn = (c == '-') ? -1 : 1;
  ret = (c == '-') ? 0 : (c - '0');
```

```
  while (c = getchar(), c >= '0' && c <= '9') ret = ret *
       10 + (c - '0');
  ret *= sgn;
  return true;
}
// 牛逼快读
namespace FastIO {
  const int MX = 4e7;
  char buf[MX];
  int c, sz;
  void Begin() {
    c = 0;
    sz = fread(buf, 1, MX, stdin);
  }
  template <class T>
  inline bool Read(T &t) {
    while (c < sz && buf[c] != '-' && (buf[c] < '0' ||
         buf[c] > '9')) c++;
    if (c >= sz) return false;
    bool flag = 0;
    if (buf[c] == '-') {
      flag = 1;
      c++;
    }
    for (t = 0; c < sz && '0' <= buf[c] && buf[c] <= '9';
         ++c) t = t * 10 + buf[c] - '0';
    if (flag) t = -t;
    return true;
  }
};
using namespace FastIO;
```

6.5 对拍

```
// windows
:loop
data.exe > in.txt
main.exe < in.txt > out.txt
std.exe < in.txt > std.txt
fc out.txt std.txt
if not errorlevel 1 goto loop
pause
:end
// Linux
declare -i n=1
while (true)
  do
  ./dtmk
  ./my < 1.in > my.out
  ./force < 1.in > for.out
  if diff my.out for.out
  then
    echo right $n
    n=n+1
  else
    exit
  fi
done
```

6.6 vimrc

```
set nu et mouse=a cin
nmap<F9> : w <cr> :!g++ % -o %< -Wall -O2 <cr> : !./%< <
     cr>
```

6.7 int128

```
using namespace std;
inline __int128 read(){
    __int128 x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-')
            f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9'){
        x=x*10+ch-'0';
        ch=getchar();
    }
    return x*f;
}
inline void print(__int128 x){
    if(x<0){
        putchar('-');
        x=-x;
    }
    if(x>9)
        print(x/10);
    putchar(x%10+'0');
}
int main(){
    __int128 a = read();
    __int128 b = read();
    print(a + b);
    cout<<endl;
    return 0;
}
```