Contents

1. Geometry

1.1 一些公式

1.1.1 Heron's Formula

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

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

1.1.2 四面体内接球球心

假设 s_i 是第 i 个顶点相对面的面积，则有

$$\begin{cases} x &= \dfrac{s_1x_1 + s_2x_2 + s_3x_3 + s_4x_4}{s_1+s_2+s_3+s_4} \\ y &= \dfrac{s_1y_1 + s_2y_2 + s_3y_3 + s_4y_4}{s_1+s_2+s_3+s_4} \\ z &= \dfrac{s_1z_1 + s_2z_2 + s_3z_3 + s_4z_4}{s_1+s_2+s_3+s_4} \end{cases}$$

体积可以使用 1/6 混合积求，内接球半径为

$$r = \frac{3V}{s_1+s_2+s_3+s_4}$$

1.1.3 三角形内心

$$\vec{I} = \frac{a\vec{A} + b\vec{B} + c\vec{C}}{a+b+c}$$

1.1.4 三角形外心

$$\vec{O} = \frac{\vec{A} + \vec{B} - \frac{\overrightarrow{BC}\cdot\overrightarrow{CA}}{\overrightarrow{AB}\times\overrightarrow{BC}}\overrightarrow{AB}^T}{2}$$

1.1.5 三角形垂心

$$\vec{H} = 3\vec{G} - 2\vec{O}$$

1.1.6 三角形偏心

$$\frac{-a\vec{A} + b\vec{B} + c\vec{C}}{-a+b+c}$$

内角的平分线和对边的两个外角平分线交点，外切圆圆心．剩余两点的同理．

1.1.7 三角形内接外接圆半径

$$r = \frac{2S}{a+b+c},\ R = \frac{abc}{4S}$$

1.1.8 Pick's Theorem 格点多边形面积

$S = I + \frac{B}{2} - 1$. I 内部点，B 边界点。

1.1.9 Euler's Formula 多面体与平面图的点、边、面

For convex polyhedron: $V - E + F = 2$.
For planar graph: $|F| = |E| - |V| + n + 1$, n : #(connected components).

1.2 三角公式

$$\sin(a \pm b) = \sin a \cos b \pm \cos a \sin b$$
$$\cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$$
$$\tan(a \pm b) = \frac{\tan(a) \pm \tan(b)}{1 \mp \tan(a)\tan(b)}$$
$$\tan(a) \pm \tan(b) = \frac{\sin(a \pm b)}{\cos(a)\cos(b)}$$
$$\sin(a) + \sin(b) = 2\sin(\frac{a+b}{2})\cos(\frac{a-b}{2})$$
$$\sin(a) - \sin(b) = 2\cos(\frac{a+b}{2})\sin(\frac{a-b}{2})$$
$$\cos(a) + \cos(b) = 2\cos(\frac{a+b}{2})\cos(\frac{a-b}{2})$$
$$\cos(a) - \cos(b) = -2\sin(\frac{a+b}{2})\sin(\frac{a-b}{2})$$
$$\sin(na) = n\cos^{n-1} a \sin a - \binom{n}{3}\cos^{n-3} a \sin^3 a + \binom{n}{5}\cos^{n-5} a \sin^5 a - \ldots$$
$$\cos(na) = \cos^n a - \binom{n}{2}\cos^{n-2} a \sin^2 a + \binom{n}{4}\cos^{n-4} a \sin^4 a - \ldots$$

1.2.1 超球坐标系

$$\begin{aligned} x_1 &= r\cos(\phi_1) \\ x_2 &= r\sin(\phi_1)\cos(\phi_2) \\ &\cdots \\ x_{n-1} &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\cos(\phi_{n-1}) \\ x_n &= r\sin(\phi_1)\cdots\sin(\phi_{n-2})\sin(\phi_{n-1}) \\ \phi_{n-1} &\in [0, 2\pi] \\ \forall i = 1..n-1\phi_i &\in [0, \pi] \end{aligned}$$

1.2.2 三维旋转公式

绕着 $(0,0,0) - (ux, uy, uz)$ 旋转 θ, (ux, uy, uz) 是单位向量

$$R = \begin{matrix} \cos\theta + u_x^2(1-\cos\theta) & u_xu_y(1-\cos\theta) - u_z\sin\theta & u_xu_z(1-\cos\theta) + u_y\sin\theta \\ u_yu_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta) - u_x\sin\theta \\ u_zu_x(1-\cos\theta) - u_y\sin\theta & u_zu_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{matrix}.$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

1.2.3 立体角公式

$$\phi : \text{二面角}$$
$$\Omega = (\phi_{ab} + \phi_{bc} + \phi_{ac})\ \mathrm{rad} - \pi\ \mathrm{sr}$$
$$\tan\left(\frac{1}{2}\Omega/\mathrm{rad}\right) = \frac{\left|\vec{a}\,\vec{b}\,\vec{c}\right|}{abc + \left(\vec{a}\cdot\vec{b}\right)c + (\vec{a}\cdot\vec{c})\,b + \left(\vec{b}\cdot\vec{c}\right)a}$$
$$\theta_s = \frac{\theta_a + \theta_b + \theta_c}{2}$$

1.2.4 常用体积公式

- 棱锥 Pyramid $V = \frac{1}{3}Sh$.
- 球 Sphere $V = \frac{4}{3}\pi R^3$.
- 棱台 Frustum $V = \frac{1}{3}h(S_1 + \sqrt{S_1S_2} + S_2)$.
- 椭球 Ellipsoid $V = \frac{4}{3}\pi abc$.
- 球缺 Spherical cap $\frac{\pi}{3}(3R - H)H^2$

1.2.5 扇形与圆弧重心

扇形重心与圆心距离为 $\frac{4r\sin(\theta/2)}{3\theta}$，圆弧重心与圆心距离为 $\frac{4r\sin^3(\theta/2)}{3(\theta - \sin(\theta))}$.

1.2.6 高维球体积

$$\begin{aligned} &V_2 = \pi R^2,\ S_2 = 2\pi R \\ &V_3 = \frac{4}{3}\pi R^3,\ S_3 = 4\pi R^2 \\ &V_4 = \frac{1}{2}\pi^2 R^4,\ S_4 = 2\pi^2 R^3 \\ &\text{Generally, } V_n = \frac{2\pi}{n}V_{n-2},\ S_{n-1} = \frac{2\pi}{n-2}S_{n-3} \\ &\text{Where, } S_0 = 2,\ V_1 = 2,\ S_1 = 2\pi,\ V_2 = \pi \end{aligned}$$

1.3 距离

欧式距离

$$\sqrt{\sum_{i=1}^{n}(x_{1,i} - x_{2,i})^2}$$

曼哈顿距离

$$\sum_{i=1}^{n}\left|x_{1,i} - x_{2,i}\right|$$

切比雪夫距离

$$\max_{i=1}^{n}\left\{\left|x_{1,i} - x_{2,i}\right|\right\}$$

曼哈顿距离与切比雪夫距离转换：

- 曼哈顿坐标系是通过切比雪夫坐标系旋转 45° 后，再缩小到原来的一半得到的。
- 将一个点 (x, y) 的坐标变为 $(x+y, x-y)$ 后，原坐标系中的曼哈顿距离等于新坐标系中的切比雪夫距离。
- 将一个点 (x, y) 的坐标变为 $(\frac{x+y}{2}, \frac{x-y}{2})$ 后，原坐标系中的切比雪夫距离等于新坐标系中的曼哈顿距离。

1.4 Pick 定理

给定顶点均为整点的简单多边形，皮克定理说明了其面积 A 和内部格点数目 i、边上格点数目 b 的关系：

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

推广：

- 取格点的组成图形的面积为一单位。在平行四边形格点，皮克定理依然成立。套用于任意三角形格点，皮克定理则是 $A = 2 \times i + b - 2$。
- 对于非简单的多边形 P，皮克定理 $A = i + \frac{b}{2} - \chi(P)$，其中 $\chi(P)$ 表示 P 的欧拉特征数 $\chi(P) = V - E + F$。
- 皮克定理和欧拉公式（$V - E + F = 2$）等价。

1.5 二维计算几何基础

```cpp
#define cp const vec &
#define cl const line &
struct vec {
    vec rot(db t) const { // 逆时针
        return {x * cos(t) - y * sin(t), x * sin(t) + y * cos(t)}; }
    vec rot90() const { return {-y, x}; }
    db len2() const { return x * x + y * y;}
    db len() const { return sqrt(x * x + y * y);}
    vec unit() const { db d = len(); return {x / d, y / d}; }
};
struct line { vec s, t; }
bool turn_left(cp a, cp b, cp c) {
 | return sgn (crs (b - a, c - a)) >= 0; }
bool point_on_segment(cp a, cl b) { // 点在线段上
    return sgn(crs(a - b.s, b.t - b.s)) == 0  // 在直线上
        && sgn(dot(b.s - a, b.t - a)) <= 0; }
```

```cpp
bool two_side(cp a, cp b, cl c) {
    return sgn(crs(a - c.s, c.t - c.s))
          * sgn(crs(b - c.s, c.t - c.s)) < 0; }
bool intersect_judge(cl a, cl b) { // 线段判非严格交
    if (point_on_segment(b.s, a)
        || point_on_segment(b.t, a)) return true;
    if (point_on_segment(a.s, b)
        || point_on_segment(a.t, b)) return true;
    return two_side(a.s, a.t, b)
          && two_side(b.s, b.t, a); }
vec line_intersect(cl a, cl b) { // 直线交点
    db s1 = crs(a.t - a.s, b.s - a.s);
    db s2 = crs(a.t - a.s, b.t - a.s);
    return (b.s * s2 - b.t * s1) / (s2 - s1); }
bool point_on_ray(cp a, cl b) {  // 点在射线上
    return sgn(crs(a - b.s, b.t - b.s)) == 0
        && sgn(dot(a - b.s, b.t - b.s)) >= 0; }
bool ray_intersect_judge(line a, line b) { // 射线判交
    db s1, s2; // can be LL
    s1 = crs(a.t - a.s, b.s - a.s);
    s2 = crs(a.t - a.s, b.t - a.s);
    if (sgn(s1) == 0 && sgn(s2) == 0) {
        return sgn(dot(a.t - a.s, b.s - a.s)) >= 0
              || sgn(dot(b.t - b.s, a.s - b.s)) >= 0; }
    if (!sgn(s1 - s2) || sgn(s1) == sgn(s2 - s1)) return 0;
    swap(a, b);
    s1 = crs(a.t - a.s, b.s - a.s);
    s2 = crs(a.t - a.s, b.t - a.s);
    return sgn(s1) != sgn(s2 - s1); }
db point_to_line(cp a, cl b) {  // 点到直线距离
    return abs(crs(b.t - b.s, a - b.s)) / dis(b.s, b.t); }
vec project_to_line(cp a, cl b) {  // 点在直线投影
    return b.s + (b.t - b.s)
          * (dot(a - b.s, b.t - b.s) / (b.t - b.s).len2()); }
db point_to_segment(cp a, cl b) {  // 点到线段距离
    if (sgn(dot(b.s - a, b.t - b.s))
        * sgn(dot(b.t - a, b.t - b.s)) <= 0)
        return abs(crs(b.t - b.s, a - b.s)) / dis(b.s, b.t);
    return min(dis(a, b.s), dis(a, b.t)); }
bool in_polygon(cp p, const vector <vec> &po) {
    int n = (int) po.size(); int cnt = 0;
    for (int i = 0; i < n; ++i) {
        vec a = po[i], b = po[(i + 1) % n];
        if (point_on_segment(p, line(a, b))) return true;
        int x = sgn(crs(p - a, b - a)),
            y = sgn(a.y - p.y), z = sgn(b.y - p.y);
        if (x > 0 && y <= 0 && z > 0) ++cnt;
        if (x < 0 && z <= 0 && y > 0) --cnt; }
    return cnt != 0; }
vector <vec> line_circle_intersect(cl a, cc b) {
    if (sgn(point_to_line(b.c, a) - b.r) > 0)
        return vector <vec> ();
    db x = sqrt(sqr(b.r) - sqr(point_to_line(b.c, a)));
    return vector <vec>
          ({project_to_line(b.c, a) + (a.s - a.t).unit() * x,
            project_to_line(b.c, a) - (a.s - a.t).unit() * x}); }
db circle_intersect_area(cc a, cc b) {
    db d = dis(a.c, b.c);
    if (sgn(d - (a.r + b.r)) >= 0) return 0;
    if (sgn(d - abs(a.r - b.r)) <= 0) {
        db r = min(a.r, b.r);
        return r * r * PI; }
    db x = (d * d + a.r * a.r - b.r * b.r) / (2 * d),
          t1 = acos(min(1., max(-1., x / a.r))),
          t2 = acos(min(1., max(-1., (d - x) / b.r)));
    return sqr(a.r) * t1 + sqr(b.r) * t2 - d * a.r * sin(t1); }
vector <vec> circle_intersect(cc a, cc b) {
    if (a.c == b.c
        || sgn(dis(a.c, b.c) - a.r - b.r) > 0
        || sgn(dis(a.c, b.c) - abs(a.r - b.r)) < 0)
        return {};
    vec r = (b.c - a.c).unit();
    db d = dis(a.c, b.c);
    db x = ((sqr(a.r) - sqr(b.r)) / d + d) / 2;
    db h = sqrt(sqr(a.r) - sqr(x));
    if (sgn(h) == 0) return {a.c + r * x};
    return {a.c + r *x + r.rot90() *h,
            a.c + r *x - r.rot90() *h}; }
// 返回按照顺时针方向
vector <vec> tangent(cp a, cc b) {
    circle p = make_circle(a, b.c);
    return circle_intersect(p, b); }
vector <line> extangent(cc a, cc b) {
    vector <line> ret;
    if (sgn(dis(a.c, b.c) - abs(a.r - b.r)) <= 0) return ret;
    if (sgn(a.r - b.r) == 0) {
        vec dir = b.c - a.c;
        dir = (dir * a.r / dir.len()).rot90();
        ret.push_back(line(a.c + dir, b.c + dir));
        ret.push_back(line(a.c - dir, b.c - dir));
    } else {
        vec p = (b.c * a.r - a.c * b.r) / (a.r - b.r);
        vector pp = tangent(p, a), qq = tangent(p, b);
        if (pp.size() == 2 && qq.size() == 2) {
            if (sgn(a.r - b.r) < 0)
                swap(pp[0], pp[1]), swap(qq[0], qq[1]);
            ret.push_back(line(pp[0], qq[0]));
            ret.push_back(line(pp[1], qq[1])); } }
    return ret; }
vector <line> intangent(cc a, cc b) {
    vector <line> ret;
    vec p = (b.c * a.r + a.c * b.r) / (a.r + b.r);
    vector pp = tangent(p, a), qq = tangent(p, b);
    if (pp.size() == 2 && qq.size() == 2) {
        ret.push_back(line(pp[0], qq[0]));
        ret.push_back(line(pp[1], qq[1])); }
    return ret; }
vector<vec> cut(const vector<vec> &c, line p) {
    vector <vec> ret;
    if (c.empty()) return ret;
    for (int i = 0; i < (int) c.size(); ++i) {
        int j = (i + 1) % (int) c.size();
        if (turn_left(p.s, p.t, c[i])) ret.push_back(c[i]);
        if (two_side(c[i], c[j], p))
            ret.push_back(line_intersect(p, line(c[i], c[j]))); }
    return ret; }
```

1.6 三角形

```cpp
vec incenter(cp a, cp b, cp c) { // 内心
    db p = dis(a, b) + dis(b, c) + dis(c, a);
    return (a * dis(b, c) + b * dis(c, a) + c * dis(a, b)) / p; }
vec circumcenter(cp a, cp b, cp c) { // 外心
    vec p = b - a, q = c - a, s(dot(p, p) / 2, dot(q, q) / 2);
    db d = crs(p, q);
    return a + vec(crs(s, vec(p.y, q.y)), crs(vec(p.x, q.x), s)) / d; }
vec orthocenter(cp a, cp b, cp c) { // 垂心
    return a + b + c - circumcenter(a, b, c) * 2.0; }
vec fermat_point(cp a, cp b, cp c) { // 费马点
    if (a == b) return a;
    if (b == c) return b;
    if (c == a) return c;
    db ab = dis(a, b), bc = dis(b, c), ca = dis(c, a);
    db cosa = dot(b - a, c - a) / ab / ca;
    db cosb = dot(a - b, c - b) / ab / bc;
    db cosc = dot(b - c, a - c) / ca / bc;
    db sq3 = PI / 3.0; vec mid;
    if (sgn(cosa + 0.5) < 0) mid = a;
    else if (sgn(cosb + 0.5) < 0) mid = b;
    else if (sgn(cosc + 0.5) < 0) mid = c;
    else if (sgn(crs(b - a, c - a)) < 0)
        mid = line_intersect(line(a, b + (c - b).rot(sq3)), line(b, c + (a - c).rot(sq3)));
    else
        mid = line_intersect(line(a, c + (b - c).rot(sq3)), line(c, b + (a - b).rot(sq3)));
    return mid; } // minimize(|A-x|+|B-x|+|C-x|)
```

1.7 凸包

```cpp
vector<vec> convex_hull(vector<vec> a) {
    int n = (int) a.size(), cnt = 0;
    if (n < 2) return a;
    sort(a.begin(), a.end());  // less<pair>
    vector<vec> ret;
```

```cpp
    for (int i = 0; i < n; ++i) {
        while (cnt > 1
        && turn_left(ret[cnt - 2], a[i], ret[cnt - 1]))
         --cnt, ret.pop_back();
        ++cnt, ret.push_back(a[i]); }
    int fixed = cnt;
    for (int i = n - 2; i >= 0; --i) {
        while (cnt > fixed
        && turn_left(ret[cnt - 2], a[i], ret[cnt - 1]))
         --cnt, ret.pop_back();
        ++cnt, ret.push_back(a[i]); }
    ret.pop_back(); return ret;
} // counter-clockwise
```

1.8 半平面交

```cpp
struct lin {
    vec s, e; db k;
    lin(vec _s, vec _e): s(_s), e(_e), k(atan2((e - s).y, (e - s).x)) {}
    il vec operator()() const {return e - s;}
};
vec cross(const lin &l1, const lin &l2) {return l1.s + l1() * crs(l2.s - l1.s, l2()) / crs(l1(), l2());}
bool cmpl(lin a, lin b) { // 极角排序，极角相同靠左优先
    if (cmp(a.k, b.k) == 0) return sign(crs(b.e-a.s, a())) > 0;
    return cmp(a.k, b.k) < 0;
}
bool Onright(lin a, lin b, lin c) { // a,b 交点在 c 右边
    vec p = cross(a, b);
    return sign(crs(c(), p - c.s)) <= 0;
}

void Halfplane(vector<lin> Ls, vector<vec> &res) { // 半平面交
    res.clear();
    sort(Ls.begin(), Ls.end(), cmpl);
    deque<int> q;
    for (int i = 0; i < (int)Ls.size(); ++i) {
        if (i != 0 && cmp(Ls[i].k, Ls[i - 1].k) == 0) continue;
        while (q.size() >= 2 && Onright(Ls[q[q.size() - 2]], Ls[q.back()], Ls[i])) q.pop_back();
        while (q.size() >= 2 && Onright(Ls[q.front()], Ls[q[1]], Ls[i])) q.pop_front();
        q.push_back(i);
    }
    while (q.size() >= 2 && Onright(Ls[q[q.size() - 2]], Ls[q.back()], Ls[q.front()])) q.pop_back();
    while (q.size() >= 2 && Onright(Ls[q[0]], Ls[q[1]], Ls[q.back()])) q.pop_front();
    if (q.size() >= 2) res.push_back(cross(Ls[q.back()], Ls[q.front()]));
    while (q.size() >= 2) {
        res.push_back(cross(Ls[q[0]], Ls[q[1]]));
        q.pop_front();
    }
}
```

2. Graph

2.1 图论基本知识

2.1.1 树链的交

2.1.2 带修改MST

维护少量修改的最小生成树，可以缩点缩边使暴力复杂度变低. (银川 21: 求有 16 个‘某两条边中至少选一条’的限制条件的最小生成树)

找出必须边　将修改边标 $-\infty$, 在MST上的其余边为必须边, 以此缩点.

找出无用边　将修改边标 ∞, 不在MST上的其余边为无用边, 删除之.

假设修改边数为 k, 操作后图中最多剩下 $k+1$ 个点和 $2k$ 条边.

2.1.3 差分约束

$x_r - x_l \le c$:`add(l, r, c)` $x_r - x_l \ge c$:`add(r, l, -c)`

2.1.4 李超线段树

添加若干条线段或直线 $(a_i, b_i) \to (a_j, b_j)$, 每次求 $[l, r]$ 上最上面的那条线段的值. 思想是让线段树中一个节点只对应一条直线, 如果在这个区间加入一条直线, 如果一段比原来的优, 一段比原来的劣, 那么判断一下两条线的交点, 判断哪条直线可以完全覆盖一段一半的区间, 把它保留, 另一条直线下传到另一半区间. 时间复杂度 $O(n \log n)$.

2.1.5 Segment Tree Beats

区间 min, max, 区间求和. 以区间取 min 为例, 额外维护最大值 m, 严格次大值 s 以及最大值个数 t. 现在假设我们要让区间 $[L, R]$ 对 x 取 min, 先在线段树中定位若干个节点, 对于每个节点分三种情况讨论: 1, 当 $m \le x$ 时, 显然这一次修改不会对这个节点产生影响, 直接退出; 2, 当 $se < x < ma$ 时, 显然这一次修改只会影响到所有最大值, 所以把 num 加上 $t*(x-ma)$, 把 ma 更新为 x, 打上标记退出; 3, 当 $se \ge x$ 时, 无法直接更新着一个节点的信息, 对当前节点的左儿子和右儿子递归处理. 单次操作均摊复杂度 $O(\log^2 n)$.

2.1.6 二分图

最小点覆盖=最大匹配数. 独立集与覆盖集互补. 最小点覆盖构造方法: 对二分图流图求割集, 跨过的边指示最小点覆盖. **Hall**定理 $G = (X, Y, E), |M| = |X| \Leftrightarrow \forall S \subseteq X, |S| \le |A(S)|$.

2.1.7 稳定婚姻问题

男士按自己喜欢程度从高到底依次向每位女士求婚, 女士遇到更喜欢的男士时就接受他, 并抛弃以前的配偶. 被抛弃的男士继续按照列表向剩下的女士依次求婚, 直到所有人都有配偶. 算法一定能得到一个匹配, 而且这个匹配一定是稳定的. 时间复杂度 $O(n^2)$.

2.1.8 三元环

对于无向边 (u, v), 如果 $\deg_u < \deg_v$, 那么连有向边 (u, v)(以点标号为第二关键字). 枚举 x 暴力即可. 时间复杂度 $O(m\sqrt{m})$.

2.1.9 图同构

令 $F_t(i) = (F_{t-1}(i) * A + \sum_{i\to j} F_{t-1}(j) * B + \sum_{j \to i} F_{t-1}(j) * C + D * (i - a)) \bmod P$, 枚举点 a, 迭代 K 次后求得的就是 a 点所对应的 $hash$ 值, 其中 K, A, B, C, D, P 为 $hash$ 参数, 可自选.

2.1.10 竞赛图 Landau's Theorem

n 个点竞赛图点按出度按升序排序, 前 i 个点的出度之和不小于 $\frac{i(i-1)}{2}$, 度数总和等于 $\frac{n(n-1)}{2}$. 否则可以用优先队列构造出方案.

2.1.11 Ramsey Theorem R(3,3)=6, R(4,4)=18

6 个人中存在 3 人相互认识或者相互不认识.

2.1.12 树的计数 Prufer序列

树和其prufer编码一一对应, 一颗 n 个点的树, 其prufer编码长度为 $n-2$, 且度数为 d_i 的点在prufer 编码中出现 $d_i - 1$ 次.

由树得到序列: 总共需要 $n-2$ 步, 第 i 步在当前的树中寻找具有最小标号的叶子节点, 将与其相连的点的标号设为Prufer序列的第 i 个元素 p_i, 并将此叶子节点从树中删除, 直到最后得到一个长度为 $n-2$ 的Prufer 序列和一个只有两个节点的树.

由序列得到树: 先将所有点的度赋初值为 1, 然后加上它的编号在Prufer序列中出现的次数, 得到每个点的度; 执行 $n-2$ 步, 第 i 步选取具有最小标号的度为 1 的点 u 与 $v = p_i$ 相连, 得到树中的一条边, 并将 u 和 v 的度减一. 最后再把剩下的两个度为 1 的点连边, 加入到树中.

相关结论: n 个点完全图, 每个点度数依次为 $d_1, d_2, \ldots, d_n$, 这样生成树的棵树为: $\frac{(n-2)!}{(d_1-1)!(d_2-1)!\ldots(d_n-1)!}$.

左边有 n_1 个点, 右边有 n_2 个点的完全二分图的生成树棵树为 $n_1^{n_2-1} \times n_2^{n_1-1}$.

m 个连通块, 每个连通块有 c_i 个点, 把他们全部连通的生成树方案数: $(\sum c_i)^{m-2} \prod c_i$

2.1.13 有根树的计数

首先, 令 $S_{n,j} = \sum_{1 \le j \le n/j}$; 于是 $n+1$ 个结点的有根树的总数为 $a_{n+1} = \frac{\sum_{j=1}^{n} j a_j S_{n-j}}{n}$. 注: $a_1 = 1, a_2 = 1, a_3 = 2, a_4 = 4, a_5 = 9, a_6 = 20, a_9 = 286, a_{11} = 1842$.

2.1.14 无根树的计数

n 是奇数时, 有 $a_n - \sum_i^{n/2} a_i a_{n-i}$ 种不同的无根树.

n 时偶数时, 有 $a_n - \sum_i^{n/2} a_i a_{n-i} + \frac{1}{2} a_{n/2}(a_{n/2} + 1)$ 种不同的无根树.

2.1.15 生成树计数 Kirchhoff's Matrix-Tree Thoerem

Kirchhoff Matrix $T = Deg - A$, Deg 是度数对角阵, A 是邻接矩阵. 无向图度数矩阵是每个点度数; 有向图度数矩阵是每个点入度.

邻接矩阵 $A[u][v]$ 表示 $u \to v$ 边个数, 重边按照边数计算, 自环不计入度数.

无向图生成树计数: $c = |K$ 的任意1个 $n-1$ 阶主子式 $|$

有向图外向树计数: $c = |$ 去掉根所在的那阶得到的主子式 $|$

2.1.16 有向图欧拉回路计数 BEST Thoerem

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

其中 deg 为入度 (欧拉图中等于出度), $t_w(G)$ 为以 w 为根的外向树的个数. 相关计算参考生成树计数.

欧拉连通图中任意两点外向树个数相同: $\mathrm{t_v}(G) = \mathrm{t_w}(G)$.

2.1.17 Tutte Matrix

Tutte matrix A of a graph $G = (V, E)$:

$$A_{ij} = \begin{cases} x_{ij} & \text{if } (i, j) \in E \text{ and } i < j \\ -x_{ij} & \text{if } (i, j) \in E \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

where x_{ij} are indeterminates. The determinant of this skew-symmetric matrix is then a polynomial (in the variables x_{ij}, $i < j$): this coincides with the square of the pfaffian of the matrix A and is non-zero (as a polynomial) if and only if a perfect matching exists.

2.1.18 Edmonds Matrix

Edmonds matrix A of a balanced ($|U| = |V|$) bipartite graph $G = (U, V, E)$:

$$A_{ij} = \begin{cases} x_{ij} & (u_i, v_j) \in E \\ 0 & (u_i, v_j) \notin E \end{cases}$$

where the x_{ij} are indeterminates. G 有完美匹配当且仅当关于 x_{ij} 的多项式 $det(A_{ij})$ 不恒为 0. 完美匹配的个数等于多项式中单项式的个数.

2.1.19 有向图无环定向, 色多项式

图的色多项式 $P_G(q)$ 对图 G 的 q-染色计数.

Triangle K_3 : $x(x-1)(x-2)$

Complete graph K_n : $x(x-1)(x-2)\cdots(x-(n-1))$

Tree with n vertices : $x(x-1)^{n-1}$

Cycle C_n : $(x-1)^n + (-1)^n(x-1)$

acyclic orientations of an n-vertex graph G is $(-1)^n P_G(-1)$.

2.2 2 SAT

```cpp
int n, m, h[2000001], p, tot, t;
int ans[2000001], cnt, bel[2000001], dfn[2000001],
  ↪ low[2000001], inz[2000001], z[2000001];
char c[2000001];
struct pp {
    int to, ne;
} b[2000001];
void add(int x, int y) {
    b[++p].to = y;
    b[p].ne = h[x];
    h[x] = p;
}
void tarjan(int x) {
    dfn[x] = low[x] = ++cnt;
    z[++t] = x; inz[x] = 1;
    for (int i = h[x]; i; i = b[i].ne) {
        int v = b[i].to;
        if (!dfn[v]) {
            tarjan(v);
            low[x] = min(low[x], low[v]);
        } else if (inz[v]) {
            low[x] = min(low[x], dfn[v]);
        }
    }
    if (dfn[x] == low[x]) {
        tot++;
        do {
            bel[z[t]] = tot;
            inz[z[t]] = 0;
        } while (z[t--] != x);
    }
}
bool solve() {
    for (int i = 1; i <= 2 * n; i++)
        if (!dfn[i]) tarjan(i);
    for (int i = 1; i <= n; i++)
        if (bel[i] == bel[i + n]) return 0;
    return 1;
}
int main() {
    scanf("%d%d", &n, &m); p = 0;
    for (int i = 1; i <= m; i++) {
```

```cpp
        int xx, yy, x, y;
        scanf("%d%d%d%d", &xx, &x, &yy, &y);
        add(xx + (x ^ 1)*n, yy + y * n);
        add(yy + (y ^ 1)*n, xx + x * n);
    }
    if (solve()) {
        printf("POSSIBLE\n");
        for (int i = 1; i <= n; i++) {
            if (bel[i] < bel[i + n]) printf("0 ");
            else printf("1 ");
        }
    } else printf("IMPOSSIBLE\n");
    return 0;
}
```

2.3 极大团

```cpp
const int maxn = 129;
int n, m, S;
int some[maxn][maxn], all[maxn][maxn], none[maxn][maxn], g[maxn][maxn];
void dfs(int d, int an, int sn, int nn) {
    if (!sn && !nn) ++S;
    if (S > 1000) return; //题意表明S超过1000就输出Impossible
    int u = some[d][0];
    for (int i = 0; i < sn; ++i) {
        int v = some[d][i];
        if (g[u][v]) continue;
        int tsn = 0, tnn = 0;
        //for(int j=0; j<an; ++j) all[d+1][j]=all[d][j];
        //all[d+1][an]=v;  //可以不写这两行
        for (int j = 0; j < sn; ++j) if (g[v][some[d][j]])
                some[d + 1][tsn++] = some[d][j];
        for (int j = 0; j < nn; ++j) if (g[v][none[d][j]])
                none[d + 1][tnn++] = none[d][j];
        dfs(d + 1, an + 1, tsn, tnn);
        some[d][i] = 0, none[d][nn++] = v;
    }
}

int main() {
    ios::sync_with_stdio(false);
    while (cin >> n >> m) {
        S = 0;
        memset(g, 0, sizeof(g));
        for (int i = 0; i < m; ++i) {
            int a, b;
            cin >> a >> b;
            g[a][b] = g[b][a] = 1;
        }
        for (int i = 0; i < n; ++i) some[0][i] = i + 1; //some的初始化
        dfs(0, 0, n, 0);
        if (S > 1000) cout << "Too many maximal sets of friends." << endl;
        else cout << S << endl;
    }
}
```

2.4 k短路

```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
const int maxn = 200010;
int n, m, s, t, k, x, y, ww, cnt, fa[maxn];

struct Edge {
  int cur, h[maxn], nxt[maxn], p[maxn], w[maxn];

  void add_edge(int x, int y, int z) {
    cur++;
    nxt[cur] = h[x];
    h[x] = cur;
    p[cur] = y;
    w[cur] = z;
  }
} e1, e2;

int dist[maxn];
bool tf[maxn], vis[maxn], ontree[maxn];

struct node {
  int x, v;

  node* operator=(node a) {
    x = a.x;
    v = a.v;
    return this;
  }

  bool operator<(node a) const { return v > a.v; }
} a;

priority_queue<node> Q;

void dfs(int x) {
  vis[x] = true;
  for (int j = e2.h[x]; j; j = e2.nxt[j])
    if (!vis[e2.p[j]])
      if (dist[e2.p[j]] == dist[x] + e2.w[j])
        fa[e2.p[j]] = x, ontree[j] = true, dfs(e2.p[j]);
}

struct LeftistTree {
  int cnt, rt[maxn], lc[maxn * 20], rc[maxn * 20], dist[maxn * 20];
  node v[maxn * 20];

  LeftistTree() { dist[0] = -1; }

  int newnode(node w) {
    cnt++;
    v[cnt] = w;
    return cnt;
  }

  int merge(int x, int y) {
    if (!x || !y) return x + y;
    if (v[x] < v[y]) swap(x, y);
    int p = ++cnt;
    lc[p] = lc[x];
    v[p] = v[x];
    rc[p] = merge(rc[x], y);
    if (dist[lc[p]] < dist[rc[p]]) swap(lc[p], rc[p]);
    dist[p] = dist[rc[p]] + 1;
    return p;
  }
} st;

void dfs2(int x) {
  vis[x] = true;
  if (fa[x]) st.rt[x] = st.merge(st.rt[x], st.rt[fa[x]]);
  for (int j = e2.h[x]; j; j = e2.nxt[j])
    if (fa[e2.p[j]] == x && !vis[e2.p[j]]) dfs2(e2.p[j]);
}

int main() {
  scanf("%d%d%d%d%d", &n, &m, &s, &t, &k);
  for (int i = 1; i <= m; i++)
    scanf("%d%d%d", &x, &y, &ww), e1.add_edge(x, y, ww), e2.add_edge(y, x, ww);
  Q.push({t, 0});
  while (!Q.empty()) {
    a = Q.top();
    Q.pop();
    if (tf[a.x]) continue;
    tf[a.x] = true;
    dist[a.x] = a.v;
    for (int j = e2.h[a.x]; j; j = e2.nxt[j])
      Q.push({e2.p[j], a.v + e2.w[j]});
  }
  if (k == 1) {
    if (tf[s])
      printf("%d\n", dist[s]);
    else
      printf("-1\n");
    return 0;
  }
  dfs(t);
  for (int i = 1; i <= n; i++)
    if (tf[i])
```

```
      for (int j = e1.h[i]; j; j = e1.nxt[j])
        if (!ontree[j])
          if (tf[e1.p[j]])
            st.rt[i] = st.merge(
                st.rt[i],
                st.newnode({e1.p[j], dist[e1.p[j]] + e1.w[j] - dist[i]}));
  for (int i = 1; i <= n; i++) vis[i] = false;
  dfs2(t);
  if (st.rt[s]) Q.push({st.rt[s], dist[s] + st.v[st.rt[s]].v});
  while (!Q.empty()) {
    a = Q.top();
    Q.pop();
    cnt++;
    if (cnt == k - 1) {
      printf("%d\n", a.v);
      return 0;
    }
    if (st.lc[a.x])  // 可并堆删除直接把左右儿子加入优先队列中
      Q.push({st.lc[a.x], a.v - st.v[a.x].v + st.v[st.lc[a.x]].v});
    if (st.rc[a.x])
      Q.push({st.rc[a.x], a.v - st.v[a.x].v + st.v[st.rc[a.x]].v});
    x = st.rt[st.v[a.x].x];
    if (x) Q.push({x, a.v + st.v[x].v});
  }
  printf("-1\n");
  return 0;
}
```

2.5 KM

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<cmath>
#include<queue>
using namespace std;
int n, N, M, k, d[501][501], match[501], ka[501], kb[501], visb[501], visa[501], p[501];
long long c[501], delta;
void bfs(int x) {
    int a, y = 0, yy = 0;
    for (int i = 1; i <= n; i++)p[i] = 0, c[i] = 1e18;
    match[y] = x;
    do {
        a = match[y], delta = 1e18, visb[y] = 1;
        for (int b = 1; b <= n; b++) {
            if (!visb[b]) {
                if (c[b] > ka[a] + kb[b] - d[a][b])
                    c[b] = ka[a] + kb[b] - d[a][b], p[b] = y;
                if (c[b] < delta)
                    delta = c[b], yy = b;
            }
        }
        for (int b = 0; b <= n; b++) {
            if (visb[b]) {
                ka[match[b]] -= delta, kb[b] += delta;
            } else c[b] -= delta;
        }
        y = yy;
    } while (match[y]);
    while (y)match[y] = match[p[y]], y = p[y];
}
long long KM() {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) visb[j] = 0;
        bfs(i);
    }
    long long ans = 0;
    for (int i = 1; i <= n; i++) ans += d[match[i]][i];
    return ans;
}
int main() {
    scanf("%d%d%d", &N, &M, &k);
    n = max(N, M);
    while (k--) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        d[y][x] = z;
    }
    printf("%lld\n", KM());
    for (int i = 1; i <= N; i++)
        printf("%d ", (d[match[i]][i] == 0) ? 0 : match[i]);
    return 0;
}
```

2.6 tarjan

```
// 强连通分量
void tarjan(int x) {
    dfn[x] = low[x] = ++tot;
    s[++len] = x;
    instack[x] = 1;
    for (int i = head[x]; i; i = e[i].next) {
        int y = e[i].to;
        if (!dfn[y]) {
            tarjan(y);
            low[x] = min(low[x], low[y]);
        } else {
            if (instack[y])low[x] = min(low[x], low[y]);
        }
    }
    if (dfn[x] == low[x]) {
        cnt++;
        ans[cnt].push_back(x);
        while (s[len] != x) {
            belong[s[len]] = cnt;
            instack[s[len]] = 0;
            ans[cnt].push_back(s[len]);
            len--;
        }
        len--;
        instack[x] = 0;
        belong[x] = cnt;
    }
}
// 边双
void tarjan(int x, int las) {
    low[x] = dfn[x] = ++cnt;
    st.push(x);
    for (auto i : e[x]) {
        if (i == las) continue;
        if (!dfn[i]) {
            tarjan(i, x);
            low[x] = min(low[x], low[i]);
        } else low[x] = min(low[x], dfn[i]);
    }
    if (dfn[x] == low[x]) {
        vector<int> vec;
        vec.push_back(x);
        while (st.top() != x) {
            vec.push_back(st.top());
            st.pop();
        }
        st.pop();
        ans.push_back(vec);
    }
}
// 点双
void tarjan(int x, int root) { //求割点的改版（其实不需要root）
    dfn[x] = low[x] = ++cnt;
    if (x == root && !head[x]) { //孤立点判定
        dcc[++ans].push_back(x);
    }
    sta.push(x);
    for (int i = head[x]; i; i = nxt[i]) {
        int g = go[i];
        if (!dfn[g]) {
            tarjan(g, root);
            low[x] = min(low[x], low[g]);
            if (low[g] >= dfn[x]) {
                ans++;
                int p;
                do { //弹栈
                    p = sta.top();
                    sta.pop();
                    dcc[ans].push_back(p);
                } while (p != g); //注意此处，因为要求是不到达出点
```

```
                dcc[ans].push_back(x);//别忘了加入源点!
            }
        } else
            low[x] = min(low[x], dfn[g]);
    }
}
```

2.7 最小斯坦纳树

```
//给定一个带边权的无向连通图G, 再给定包含k个结点的点集S, 选
  ↪ 出G的子图G', 使得G' 包含S, G' 为连通图, 且G' 边权和最小
#include<bits/stdc++.h>
#define mp make_pair
#define zjx printf("%d",
#define AK dp[c[1]][(1<<k)-1]
#define IOI );
using namespace std;
int n, m, k, p, h[101], dp[101][1024], c[11], vis[101];
struct tree {
    int to, ne, v;
} a[1001];
void add(int x, int y, int z) {
    a[++p].to = y;
    a[p].ne = h[x];
    a[p].v = z;
    h[x] = p;
}
priority_queue<pair<int, int>, vector<pair<int, int> >,
  ↪ greater<pair<int, int> > >q;
void dijkstra(int s) {
    memset(vis, 0, sizeof(vis));
    while (!q.empty()) {
        int x = q.top().second;
        q.pop();
        if (vis[x])continue;
        vis[x] = 1;
        for (int i = h[x]; i; i = a[i].ne) {
            if (dp[a[i].to][s] > dp[x][s] + a[i].v) {
                dp[a[i].to][s] = dp[x][s] + a[i].v;
                q.push(mp(dp[a[i].to][s], a[i].to));
            }
        }
    }
}
int main() {
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 1; i <= m; i++) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        add(x, y, z), add(y, x, z);
    }
    memset(dp, 0x3f, sizeof(dp));
    for (int i = 1; i <= k; i++)scanf("%d", &c[i]),
      ↪ dp[c[i]][1 << (i - 1)] = 0;
    for (int s = 1; s < (1 << k); s++) {
        for (int i = 1; i <= n; i++) {
            for (int ss = s & (s - 1); ss; ss = (ss - 1)&s)
                dp[i][s] = min(dp[i][s], dp[i][ss] + dp[i]
                  ↪ [ss ^ s]);
            if (dp[i][s] != 0x3f3f3f3f)q.push(mp(dp[i][s],
              ↪ i));
        }
        dijkstra(s);
    }
    zjx AK IOI
    return 0;
}
```

3. Data Structure

3.1 LCT 动态树

```cpp
namespace LCT {
    int ch[N][2], f[N], sum[N], val[N], tag[N], dat[N]; // dat 维护的链信息, val 点上信息
    inline void PushUp(int x) {
        dat[x] = dat[ch[x][0]] ^ dat[ch[x][1]] ^ val[x];
    }
    inline void PushRev(int x) {swap(ch[x][0], ch[x][1]); tag[x] ^= 1;}
    inline void PushDown(int x) {
        if (tag[x] == 0) return ;
        PushRev(ch[x][0]); PushRev(ch[x][1]); tag[x] = 0;
    }
    inline bool Get(int x) {return ch[f[x]][1] == x;} // 是父亲的哪个儿子
    inline bool IsRoot(int x) {return (ch[f[x]][1] != x && ch[f[x]][0] != x);} // 是否是当前 Splay 的根
    inline void Rotate(int x) { // Splay 旋转
        int y = f[x], z = f[y], k = Get(x);
        if (!IsRoot(y)) ch[z][Get(y)] = x;
        ch[y][k] = ch[x][k ^ 1]; f[ch[x][k ^ 1]] = y;
        ch[x][k ^ 1] = y; f[y] = x; f[x] = z;
        PushUp(y); PushUp(x);
    }
    void Updata(int x) { // Splay 中从上到下 PushDown
        if (!IsRoot(x)) Updata(f[x]);
        PushDown(x);
    }
    inline void Splay(int x) { // Splay 上把 x 转到根
        Updata(x);
        for (int fa; fa = f[x], !IsRoot(x); Rotate(x)) {
            if (!IsRoot(fa)) Rotate(Get(fa) == Get(x) ? fa : x);
        }
        PushUp(x);
    }
    inline void Access(int x) { // 辅助树上打通 x 到根的路径(即 x 到根变为实链)
        for (int p = 0; x; p = x, x = f[x]) {
            Splay(x); ch[x][1] = p; PushUp(x);
        }
    }
    inline void MakeRoot(int x) { // 钦定 x 为辅助树根
        Access(x); Splay(x); PushRev(x);
    }
    inline int FindRoot(int x) { // 找 x 所在辅助树根
        Access(x); Splay(x);
        while (ch[x][0]) PushDown(x), x = ch[x][0];
        Splay(x); // 不加复杂度会假
        return x;
    }
    inline void Split(int x, int y) { // 把 x 到 y 的路径提出来, 并以 y 为 Splay 根
        MakeRoot(x); Access(y); Splay(y);
    }
    inline bool Link(int x, int y) { // 连接 x,y 两点
        MakeRoot(x);
        if (FindRoot(y) == x) return false;
        f[x] = y;
        return true;
    }
    inline bool Cut(int x, int y) { // x,y 断边
        MakeRoot(x);
        if (FindRoot(y) == x && f[y] == x && !ch[y][0]) {
            f[y] = ch[x][1] = 0; PushUp(x);
            return true;
        }
        return false;
    }
}
```

3.2 KD Tree

```cpp
// KDTree 二维平面邻域查询 K 远点对 n=1e5 k=100
priority_queue<ll, vector<ll>, greater<ll> >q; // 小根堆
namespace KDTree {
    struct node {
        int X[2];
        int &operator[](const int k) {return X[k];}
    } p[N];
    int nowd;
    bool cmp(node a, node b) {return a.X[nowd] < b.X[nowd];}
    int lc[N], rc[N], L[N][2], R[N][2]; // lc/rc 左右孩子; L/R 对应超矩形各个维度范围
    inline ll sqr(int x) {return 1ll * x * x;}
    void pushup(int x) { // 更新该点所代表空间范围
        L[x][0] = R[x][0] = p[x][0];
        L[x][1] = R[x][1] = p[x][1];
        if (lc[x]) {
            umin(L[x][0], L[lc[x]][0]); umax(R[x][0], R[lc[x]][0]);
            umin(L[x][1], L[lc[x]][1]); umax(R[x][1], R[lc[x]][1]);
        }
        if (rc[x]) {
            umin(L[x][0], L[rc[x]][0]); umax(R[x][0], R[rc[x]][0]);
            umin(L[x][1], L[rc[x]][1]); umax(R[x][1], R[rc[x]][1]);
        }
    }
    int build(int l, int r) {
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        // >>> 方差建树
        db av[2] = {0, 0}, va[2] = {0, 0}; // av 平均数, va 方差
        for (int i = l; i <= r; ++i) av[0] += p[i][0], av[1] += p[i][1];
        av[0] /= (r - l + 1); av[1] /= (r - l + 1);
        for (int i = l; i <= r; ++i) {
            va[0] += sqr(av[0] - p[i][0]);
            va[1] += sqr(av[1] - p[i][1]);
        }
        if (va[0] > va[1]) nowd = 0;
        else nowd = 1; // 找方差大的维度划分
        // >>> 轮换建树 nowd=dep%D
        nth_element(p + l, p + mid, p + r + 1, cmp); // 以该维度中位数分割
        lc[mid] = build(l, mid - 1); rc[mid] = build(mid + 1, r);
        pushup(mid);
        return mid;
    }
    ll dist(int a, int x) { // 估价函数, 点 a 到树上 x 点对应空间最远距离
        return max(sqr(p[a][0] - L[x][0]), sqr(p[a][0] - R[x][0])) +
               max(sqr(p[a][1] - L[x][1]), sqr(p[a][1] - R[x][1]));
    }
    void query(int l, int r, int a) { // 点 a 邻域查询
        if (l > r) return ;
        int mid = (l + r) >> 1;
        ll t = sqr(p[mid][0] - p[a][0]) + sqr(p[mid][1] - p[a][1]);
        if (t > q.top()) q.pop(), q.push(t); // 更新答案
        ll disl = dist(a, lc[mid]), disr = dist(a, rc[mid]);
        if (disl > q.top() && disr > q.top()) // 两边都有机会更新, 优先搜大的
            (disl > disr) ? (query(l, mid - 1, a), query(mid + 1, r, a)) : (query(mid + 1, r, a), query(l, mid - 1, a));
        else
            (disl > q.top()) ? query(l, mid - 1, a) : query(mid + 1, r, a);
    }
}
using namespace KDTree;
int main() {
    red(n); red(k); k *= 2;
    for (int i = 1; i <= k; ++i) q.push(0);
    for (int i = 1; i <= n; ++i) red(p[i][0]), red(p[i][1]);
    build(1, n);
    for (int i = 1; i <= n; ++i) query(1, n, i);
    printf("%lld\n", q.top());
}
```

```cpp
// 动态 KDTree 维护空间权值 (单点修改 & 空间查询)
// 时间复杂度 O(log n) ~ O(n^(1-1/k))
```

```cpp
#define sqr(x) ((x) * (x))
namespace KDT {
    struct dat {
        int X[2];
        int &operator[](const int k) {return X[k];}
    } p[N];
    db alp = 0.725; // 重构常数
    int nowd;
    bool cmp(int a, int b) {return p[a][nowd] < p[b][nowd];}
    //  root: 根 cur: 总点数 d: 当前分割维度 lc/rc: 左右儿子 L/R: 当前空间范围 siz: 子树大小  sum/val 空间的值，单点的值
    int root, cur, d[N], lc[N], rc[N], L[N][2], R[N][2], siz[N], sum[N], val[N];
    int g[N], t; // 用于重构的临时数组
    void pushup(int x) {
        siz[x] = siz[lc[x]] + siz[rc[x]] + 1;
        sum[x] = sum[lc[x]] + sum[rc[x]] + val[x];
        L[x][0] = R[x][0] = p[x][0];
        L[x][1] = R[x][1] = p[x][1];
        if (lc[x]) {
            umin(L[x][0], L[lc[x]][0]); umax(R[x][0], R[lc[x]][0]);
            umin(L[x][1], L[lc[x]][1]); umax(R[x][1], R[lc[x]][1]);
        }
        if (rc[x]) {
            umin(L[x][0], L[rc[x]][0]); umax(R[x][0], R[rc[x]][0]);
            umin(L[x][1], L[rc[x]][1]); umax(R[x][1], R[rc[x]][1]);
        }
    }
    int build(int l, int r) { // 对 g[1...t] 进行建树 , 对应点都是 g[x]。方差建树
        if (l > r) return 0;
        int mid = (l + r) >> 1;
        db av[2] = {0, 0}, va[2] = {0, 0};
        for (int i = l; i <= r; ++i) av[0] += p[g[i]][0], av[1] += p[g[i]][1];
        av[0] /= (r - l + 1); av[1] /= (r - l + 1);
        for (int i = l; i <= r; ++i) va[0] += sqr(av[0] - p[g[i]][0]), va[1] += sqr(av[1] - p[g[i]][1]);
        if (va[0] > va[1]) d[g[mid]] = nowd = 0;
        else d[g[mid]] = nowd = 1;
        nth_element(g + l, g + mid, g + r + 1, cmp);
        lc[g[mid]] = build(l, mid - 1); rc[g[mid]] = build(mid + 1, r);
        pushup(g[mid]);
        return g[mid];
    }
    void expand(int x) { // 将子树展开到临时数组里
        if (!x) return;
        expand(lc[x]);
        g[++t] = x;
        expand(rc[x]);
    }
    void rebuild(int &x) { // x 所在子树重构
        t = 0; expand(x);
        x = build(1, t);
    }
    bool chk(int x) {return alp * siz[x] <= (db)max(siz[lc[x]], siz[rc[x]]);} // 判断失衡
    void insert(int &x, int a) { // 插入点 a , p[a],val[a] 为其信息
        if (!x) { x = a; pushup(x); d[x] = rand() & 1; return; }
        if (p[a][d[x]] <= p[x][d[x]]) insert(lc[x], a);
        else insert(rc[x], a);
        pushup(x);
        if (chk(x)) rebuild(x); // 失衡暴力重构
    }
    dat Lt, Rt; // 询问一块空间的值 (为了减小常数把参数放在外面)
    int query(int x) {
        if (!x || Rt[0] < L[x][0] || Lt[0] > R[x][0] || Rt[1] < L[x][1]
            || Lt[1] > R[x][1]) return 0; // 结点为空或与询问区间无交
        if (Lt[0] <= L[x][0] && R[x][0] <= Rt[0] && Lt[1] <= L[x][1]
            && R[x][1] <= Rt[1]) return sum[x]; // 区间完全覆盖
        int ret = 0;
        if (Lt[0] <= p[x][0] && p[x][0] <= Rt[0] && Lt[1] <= p[x][1]
            && p[x][1] <= Rt[1]) ret += val[x]; // 当前点在区间内
        return query(lc[x]) + query(rc[x]) + ret;
    }
}
using namespace KDT;
int main() {
    int n; read(n);
    for (int op;;) {
        read(op);
        switch (op) {
        case 1:
            ++cur; read(p[cur][0]); read(p[cur][1]); read(val[cur]);
            insert(root, cur);
            break;
        case 2:
            read(Lt[0]); read(Lt[1]); read(Rt[0]); read(Rt[1]);
            printf("%d\n", query(root));
            break;
        case 3: | return 0; break;
        }
    }
    return 0;
}
```

3.3 李超线段树

```cpp
// 李超线段树 对于 (x1,y1) (x2,y2) -> y=0*x+max(y1,y2) [x1,x1]
#define ls (x<<1)
#define rs (x<<1|1)
typedef long long ll;
typedef double db;
const int N = 100010;
const int M = 40000;
struct line {
    db k, b;
} lin[N];
db val(int id, db X) {return lin[id].k * X + lin[id].b;}
int D[N << 2], n, id;
void modify(int L, int R, int id, int l = 1, int r = M - 1, int x = 1) { // 线 lin[id], 范围 [L, R]
    if (L <= l && r <= R) {
        int mid = (l + r) >> 1, lid = D[x];
        db lst = val(D[x], mid), now = val(id, mid);
        if (l == r) {if (now > lst) D[x] = id; return ;}
        if (lin[id].k > lin[D[x]].k) {
            if (now > lst) D[x] = id, modify(L, R, lid, l, mid, ls); // id->lid
            else modify(L, R, id, mid + 1, r, rs);
        } else if (lin[id].k < lin[D[x]].k) {
            if (now > lst) D[x] = id, modify(L, R, lid, mid + 1, r, rs); // id->lid
            else modify(L, R, id, l, mid, ls);
        } else if (lin[id].b > lin[D[x]].k) D[x] = id;
        return ;
    }
    int mid = (l + r) >> 1;
    if (L <= mid) modify(L, R, id, l, mid, x << 1);
    if (R > mid) modify(L, R, id, mid + 1, r, x << 1 | 1);
}
int gmax(int x, int y, int ps) {
    if (val(x, ps) > val(y, ps)) return x;
    if (val(x, ps) < val(y, ps)) return y;
    return (x < y) ? x : y;
}
int query(int ps, int l = 1, int r = M - 1, int x = 1) { // 查 x=ps
    if (l == r) return D[x];
    int mid = (l + r) >> 1, ret = D[x], t = 0;
    if (ps <= mid)
        t = query(ps, l, mid, ls);

    else
        t = query(ps, mid + 1, r, rs);
    return gmax(ret, t, ps);
```

```
}
```

3.4 吉司机线段树

```
/*
 * seg-beats 吉司机线段树
 * 区间最值操作
 * 支持 区间取min，区间取max，区间加减，区间求和，区间最小/大
 * 复杂度 O(m log n)
 */
#define ls (x << 1)
#define rs (x << 1 | 1)
#define mid ((l + r) >> 1)
typedef long long ll;
const int N = 500010;
const int inf = 0x3f3f3f3f;
struct datmn {
    int fi, se, cnt; // 最小值，次小值，最小值个数
    datmn() {fi = se = inf; cnt = 0;}
    void ins(int x, int c) {
        if (x < fi) se = fi, cnt = c, fi = x;
        else if (x == fi) cnt += c;
        else if (x < se) se = x;
    }
    friend datmn operator+(const datmn &a, const datmn &b) {
        datmn r = a; r.ins(b.fi, b.cnt); r.ins(b.se, 0); return r;
    }
};
struct datmx {
    int fi, se, cnt;
    datmx() {fi = se = -inf; cnt = 0;}
    void ins(int x, int c) {
        if (x > fi) se = fi, cnt = c, fi = x;
        else if (x == fi) cnt += c;
        else if (x > se) se = x;
    }
    friend datmx operator+(const datmx &a, const datmx &b) {
        datmx r = a; r.ins(b.fi, b.cnt); r.ins(b.se, 0); return r;
    }
};

struct node {
    datmn mn; datmx mx;
    ll sum; int addmn, addmx, add, len;
} t[N << 2];
int n, m, a[N];
void pushup(int x) {
    t[x].mx = t[ls].mx + t[rs].mx;
    t[x].mn = t[ls].mn + t[rs].mn;
    t[x].sum = t[ls].sum + t[rs].sum;
}
void build(int l = 1, int r = n, int x = 1) {
    t[x].add = t[x].addmn = t[x].addmx = 0;
    t[x].len = r - l + 1;
    if (l == r) {
        t[x].mx = datmx(); t[x].mx.ins(a[l], 1);
        t[x].mn = datmn(); t[x].mn.ins(a[l], 1);
        t[x].sum = a[l];
        return;
    }
    build(l, mid, ls); build(mid + 1, r, rs);
    pushup(x);
}
void update(int x, int vn, int vx, int v) { // vn: addmn, vx: addmx, v: add
    // 所有数相同特判，此时最大值 tag 和最小值 tag 应该相同且不等于其他值 tag
    if (t[x].mn.fi == t[x].mx.fi) {
        if (vn == v) vn = vx;
        else vx = vn;
        t[x].sum += (ll)vn * t[x].mn.cnt;
    } else t[x].sum += (ll)vn * t[x].mn.cnt + (ll) vx * t[x].mx.cnt + (ll)v * (t[x].len - t[x].mn.cnt - t[x].mx.cnt);
    if (t[x].mn.se == t[x].mx.fi) t[x].mn.se += vx; // 次小值 = 最大值，应该用最大值 tag 处理
    else if (t[x].mn.se != inf) t[x].mn.se += v;
    if (t[x].mx.se == t[x].mn.fi) t[x].mx.se += vn; // 次大值同理
    else if (t[x].mx.se != -inf) t[x].mx.se += v;
    t[x].mn.fi += vn; t[x].mx.fi += vx;
    t[x].addmn += vn; t[x].addmx += vx; t[x].add += v;
}
void pushdown(int x) {
    int mn = min(t[ls].mn.fi, t[rs].mn.fi);
    int mx = max(t[ls].mx.fi, t[rs].mx.fi);
    update(ls, (mn == t[ls].mn.fi) ? t[x].addmn : t[x].add, (mx == t[ls].mx.fi) ? t[x].addmx : t[x].add, t[x].add);
    update(rs, (mn == t[rs].mn.fi) ? t[x].addmn : t[x].add, (mx == t[rs].mx.fi) ? t[x].addmx : t[x].add, t[x].add);
    t[x].add = t[x].addmn = t[x].addmx = 0;
}
void modifyadd(int L, int R, int v, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return ;
    if (L <= l && r <= R) return update(x, v, v, v);
    pushdown(x);
    modifyadd(L, R, v, l, mid, ls);
    modifyadd(L, R, v, mid + 1, r, rs);
    pushup(x);
}
void modifymin(int L, int R, int v, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return ;
    if (L <= l && r <= R && v > t[x].mx.se) {
        if (v >= t[x].mx.fi) return ;
        update(x, 0, v - t[x].mx.fi, 0);
        return;
    }
    pushdown(x);
    modifymin(L, R, v, l, mid, ls);
    modifymin(L, R, v, mid + 1, r, rs);
    pushup(x);
}
void modifymax(int L, int R, int v, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return ;
    if (L <= l && r <= R && v < t[x].mn.se) {
        if (v <= t[x].mn.fi) return;
        update(x, v - t[x].mn.fi, 0, 0);
        return;
    }
    pushdown(x);
    modifymax(L, R, v, l, mid, ls);
    modifymax(L, R, v, mid + 1, r, rs);
    pushup(x);
}
int querymax(int L, int R, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return -inf;
    if (L <= l && r <= R) return t[x].mx.fi;
    pushdown(x);
    return max(querymax(L, R, l, mid, ls), querymax(L, R, mid + 1, r, rs));
}
int querymin(int L, int R, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return inf;
    if (L <= l && r <= R) return t[x].mn.fi;
    pushdown(x);
    return min(querymin(L, R, l, mid, ls), querymin(L, R, mid + 1, r, rs));
}
ll querysum(int L, int R, int l = 1, int r = n, int x = 1) {
    if (r < L || R < l) return 0;
    if (L <= l && r <= R) return t[x].sum;
    pushdown(x);
    return querysum(L, R, l, mid, ls) + querysum(L, R, mid + 1, r, rs);
}
```

3.5 FHQ Treep

```
// fhq - treap 简易模板
#define ls(p) t[p].l
#define rs(p) t[p].r
#define mid ((l+r)>>1)
using namespace std;
```

```cpp
const int N = 100010;
mt19937 rd(random_device{}());
struct node {
    int l, r, siz, rnd, val, tag;
} t[N]; int tot, root;
/* 节点回收
int cyc[N],cyccnt;
inline void delnode(int p) {cyc[++cyccnt]=p;}
inline void newnode(int val) {
    int id=(cyccnt>0)?cyc[cyccnt--]:++tot;
    t[id]={0,0,1,(int)(rd()),val}; return id;
}
*/
inline int newnode(int val) { t[++tot] = {0, 0, 1, (int)(rd()), val}; return tot;}
inline void updata(int p) {
    t[p].siz = t[ls(p)].siz + t[rs(p)].siz + 1;
    /* maintain */
}
inline void pushtag(int p, int vl) { /* tag to push */ }
inline void pushdown(int p) {
    if (t[p].tag != std_tag) {
        if (ls(p)) pushtag(ls(p), t[p].tag);
        if (rs(p)) pushtag(rs(p), t[p].tag);
        t[p].tag = std_tag;
    }
}
int merge(int p, int q) {
    if (!p || !q) return p + q;
    if (t[p].rnd < t[q].rnd) {
        pushdown(p);
        rs(p) = merge(rs(p), q);
        updata(p); return p;
    } else {
        pushdown(q);
        ls(q) = merge(p, ls(q));
        updata(q); return q;
    }
}
void split(int p, int k, int &x, int &y) {
    if (!p) x = 0, y = 0;
    else {
        pushdown(p);
        if (t[ls(p)].siz >= k) y = p, split(ls(p), k, x, ls(p));
        else x = p, split(rs(p), k - t[ls(p)].siz - 1, rs(p), y);
        updata(p);
    }
}
int build(int l, int r) { // build tree on a[l..r], return the root
    if (l > r) return 0;
    return merge(build(l, mid - 1), merge(newnode(a[mid]), build(mid + 1, r)));
}
```

3.6 哈希表

```cpp
typedef long long ll;
const int M = 19260817;
const int MAX_SIZE = 2000000;
struct Hash_map {
    struct data {
        int nxt;
        ll key, value; // (key,value)
    } e[MAX_SIZE];
    int head[M], size;
    inline int f(ll key) { return key % M; }
    ll &operator[](const ll &key) {
        int ky = f(key);
        for (int i = head[ky]; i != -1; i = e[i].nxt)
            if (e[i].key == key) return e[i].value;
        return e[++size] = data{head[ky], key, 0}, head[ky] = size, e[size].value;
    }
    void clear() {
        memset(head, -1, sizeof(head));
        size = 0;
    }
    Hash_map() {clear();}
};
```

4. String

4.1 最小表示法

```cpp
//n为串长，a下标从0开始
int Min_show(int *a, int n) {
    int i = 0, j = 1, k = 0;
    while (i < n && j < n && k < n) {
        auto u = a[(i + k) % n];
        auto v = a[(j + k) % n];
        if (u == v) ++k;
        else {
            if (u > v) i += k + 1;
            else j += k + 1;
            if (i == j) ++j;
            k = 0;
        }
    }
    return min(i, j);
}
```

4.2 AC 自动机

```cpp
int son[M][26], fail[M], cnt = 0;
void ins(const char *s) {
    int p = 0, n = strlen(s + 1);
    for (int i = 1; i <= n; ++i) {
        int c = s[i] - 'a';
        if (!son[p][c]) son[p][c] = ++cnt;
        p = son[p][c];
    }
}
queue<int> q;
void get_fail() {
    for (int c = 0; c < 26; ++c)
        if (son[0][c]) q.push(son[0][c]);
    while (!q.empty()) {
        int x = q.front(); q.pop();
        for (int c = 0; c < 26; ++c) {
            if (son[x][c]) {
                fail[son[x][c]] = son[fail[x]][c];
                q.push(son[x][c]);
            } else son[x][c] = son[fail[x]][c];
        }
    }
}

```

4.3 回文树

```cpp
int len[M], fa[M], son[M][26], lst, cnt, f[M];
char s[M];
int extend(int n) {
    int p = lst, c = s[n] - 'a';
    while (s[n - len[p] - 1] != s[n]) p = fa[p];
    if (!son[p][c]) {
        int now = p;
        len[++cnt] = len[p] + 2;//回文串长度
        p = fa[p];
        while (s[n - len[p] - 1] != s[n]) p = fa[p];
        fa[cnt] = son[p][c];
        lst = son[now][c] = cnt;
        f[cnt] = f[fa[cnt]] + 1;//回文串数量
    } else lst = son[p][c];
    return f[lst];
}
int main() {
    fa[0] = cnt = 1;
    val[1] = -1;
}
```

4.4 Manacher

```cpp
char s[M << 1];
int p[M];
//n为串长，a下标从1开始，p为回文串半径 (0~2n+1)
void Manacher(const char *a, int n) {
    int r = 0, mid;
    for (int i = 1; i <= n; ++i) s[i << 1] = a[i];
    for (int i = 0; i <= n; ++i) s[i * 2 + 1] = '#';
    s[0] = '#'; n = n << 1 | 1;
    for (int i = 1; i <= n; ++i) {
```

```cpp
        p[i] = (i <= r ? min(p[mid * 2 - i], p[mid] + mid - i) : 1);
        while (s[i - p[i] - 1] == s[i + p[i] + 1]) ++p[i];
        if (i + p[i] > r) r = i + p[i], mid = i;
    }
}
```

4.5 字符串哈希

```cpp
const int HA = 2;
const int PP[] = {318255569, 66604919, 19260817}, QQ[] = {1010451419, 1011111133, 1033111117};

int pw[HA][N];
void HashInit() {
    for (int h = 0; h < HA; h++) {
        pw[h][0] = 1;
        for (int i = 1; i < N; i++)
            pw[h][i] = (LL)pw[h][i - 1] * PP[h] % QQ[h];
    }
}
struct Hash {
    int hs[HA], len;
    Hash() {
        memset(hs, 0, sizeof hs);
        len = 0;
    }
    Hash(int x) {
        for (int h = 0; h < HA; h++) hs[h] = x;
        len = 1;
    }
    Hash operator + (const int &x)const {
        Hash res;
        res.len = len + 1;
        for (int h = 0; h < HA; h++)
            res.hs[h] = ((LL)hs[h] * PP[h] + x) % QQ[h];
        return res;
    }
    Hash operator - (const Hash &x)const {
        Hash res;
        res.len = len - x.len;
        for (int h = 0; h < HA; h++) {
            res.hs[h] = (hs[h] - (LL)pw[h][res.len] * x.hs[h]) % QQ[h];
            if (res.hs[h] < 0) res.hs[h] += QQ[h];
        }
        return res;
    }
    bool operator == (const Hash &x)const {
        for (int h = 0; h < HA; h++)
            if (hs[h] != x.hs[h]) return false;
        return len == x.len;
    }
    // below : not that frequently used
    Hash operator + (const Hash &x)const {
        Hash res;
        res.len = len + x.len;
        for (int h = 0; h < HA; h++)
            res.hs[h] = ((LL)hs[h] * pw[h][x.len] + x.hs[h]) % QQ[h];
        return res;
    }
} H;
Hash operator + (const int &a, const Hash &b) {
    Hash res;
    res.len = b.len + 1;
    for (int h = 0; h < HA; h++)
        res.hs[h] = ((LL)a * pw[h][b.len] + b.hs[h]) % QQ[h];
    return res;
}
```

4.6 SA

```cpp
// rnk: 排名, sa: 位置
// height[i] = lcp(sa[i], sa[i - 1])
// M开两倍
void get_sa(char *s, int n, int *sa, int *rnk, int *height) { // 1-based
    static int c[M], p[M], t[M];
    int m = 300;
    for (int i = 1; i <= n; ++i) ++c[p[i] = s[i]];
    for (int i = 2; i <= m; ++i) c[i] += c[i - 1];
    for (int i = n; i; --i) sa[c[p[i]]--] = i;
    for (int k = 1; k < n ; k <<= 1) {
        int cnt = 0;
        for (int i = n - k + 1; i <= n; ++i) t[++cnt] = i;
        for (int i = 1; i <= n; ++i) if (sa[i] > k) t[++cnt] = sa[i] - k;
        for (int i = 1; i <= m; ++i) c[i] = 0;
        for (int i = 1; i <= n; ++i) ++c[p[i]];
        for (int i = 2; i <= m; ++i) c[i] += c[i - 1];
        for (int i = n; i; --i) sa[c[p[t[i]]]--] = t[i], t[i] = 0;
        swap(p, t);
        p[sa[1]] = cnt = 1;
        for (int i = 2; i <= n; ++i) {
            if (t[sa[i]] != t[sa[i - 1]] || t[sa[i] + k] != t[sa[i - 1] + k]) ++cnt;
            p[sa[i]] = cnt;
        }
        if (cnt == n) break;
        m = cnt;
    }
    for (int i = 1; i <= n; i++) rnk[sa[i]] = i;
    for (int i = 1, k = 0; i <= n; i++) {
        if (k) k--;
        while (s[i + k] == s[sa[rnk[i] - 1] + k]) k++;
        height[rnk[i]] = k;
    }
}
char s[M];
int sa[M], rnk[M], height[M];
int main() {
    cin >> (s + 1);
    int n = strlen(s + 1);
    get_sa(s, n, sa, rnk, height);
    for (int i = 1; i <= n; i++)
        cout << sa[i] << (i < n ? ' ' : '\n');
    for (int i = 2; i <= n; i++)
        cout << height[i] << (i < n ? ' ' : '\n');
    return 0;
}
```

4.7 SAM

```cpp
int lst = 1, cnt = 1, len[M], fa[M], son[M][26];
void Extend(int c) { // 结点数要开成串长的两倍
    int p = lst, np = lst = ++cnt;
    len[np] = len[p] + 1;
    for (; p && !son[p][c]; p = fa[p]) son[p][c] = np;
    if (!p) return fa[lst = np] = 1, void();
    int q = son[p][c];
    if (len[q] == len[p] + 1)
        return fa[lst = np] = q, void();
    int nq = ++cnt;
    len[nq] = len[p] + 1;
    fa[nq] = fa[q];
    fa[np] = fa[q] = nq;
    memcpy(son[nq], son[q], sizeof(son[q]));
    for (; p && son[p][c] == q; p = fa[p]) son[p][c] = nq;
    lst = np;
}
int c[M], q[M];
int main() {
    for (int i = 1; i <= n ; ++i) Extend(s[i] - 'a');
    for (int i = 1; i <= cnt; i++) ++c[len[i]];
    for (int i = 1; i <= cnt; i++) c[i] += c[i - 1];
    for (int i = 1; i <= cnt; i++) q[c[len[i]]--] = i;
    return 0;
}
```

4.8 KMP and EXKMP

```cpp
// 1-based
int fail[M];
void KMP(const char *s, int n) {
    fail[0] = fail[1] = 0;
    for (int i = 2, j = 0; i <= n; i++) {
        fail[i] = 0;
        while (j && s[i] != s[j + 1]) j = fail[j];
        if (s[i] == s[j + 1]) fail[i] = ++j;
    }
}
// match
```

```cpp
for (int i = 1, j = 0; i <= la; ++i) {
    while (j && b[j + 1] != a[i]) j = fail[j];
    if (b[j + 1] == a[i]) ++j;
    if (j == lb) {
        printf("%d\n", i - lb + 1);
        j = fail[j];
    }
}
// 0-based
// s 和 s 的每一个后缀的最长公共前缀 (LCP) 长度数组
void exKMP(const char *s, int *z, int n) {// get z
    int l = 0, r = 0;
    z[0] = n;
    for (int i = 1; i <= n; ++i) {
        z[i] = i > r ? 0 : min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) r = i + z[l = i] - 1;
    }
}
// t 与 s 的每一个后缀的 LCP 长度数组
void exKMP(const char *s, const char *t, int *z, int *p, int sn) {// get p
    int l = -1, r = -1;
    for (int i = 0; i <= sn; ++i) {
        p[i] = i > r ? 0 : min(r - i + 1, z[i - l]);
        while (i + p[i] < sn && t[p[i]] == s[i + p[i]]) ++p[i];
        if (i + p[i] - 1 > r) r = i + p[l = i] - 1;
    }
}
```

4.9 Lydon

```cpp
/*
满足s的最小后缀等于s本身的串s称为Lyndon串.
等价于: s是它自己的所有循环移位中唯一最小的一个.
任意字符串s可以分解为 s = s1s2...sk , 其中 si 是Lyndon串,
si ≥si+1. 且这种分解方法是唯一的.
*/
void mnsuf(char *s, int *mn, int n) { // 每个前缀的最小后缀
 | for (int i = 0; i < n; ) {
 | | int j = i, k = i + 1;
 | | mn[i] = i;
 | | for (; k < n && s[j] <= s[k]; ++ k)
 | | | if (s[j] == s[k]) mn[k] = mn[j] + k - j, ++j;
 | | | else mn[k] = j = i;
 | | while(i <= j)i += k - j;
 | }
} // lyn+=s[i..i+kj1]
void mxsuf(char *s, int *mx, int n) { // 每个前缀的最大后缀
 | fill(mx, mx + n, -1);
 | for (int i = 0; i < n; ) {
 | | int j = i, k = i + 1;
 | | if (mx[i] == -1) mx[i] = i;
 | | for (; k < n && s[j] >= s[k]; ++k) {
 | | | j = s[j] == s[k] ? j + 1 : i;
 | | | if (mx[k] == -1) mx[k] = i;
 | | }
 | | while(i <= j)i += k - j;
 | }
}
```

4.10 SASAM后缀树

```cpp
const int M=1e5;
bool vis[M << 1];
char s[M];
int id[M << 1], ch[M << 1][26], height[M], tim = 0;
void dfs(int x) {
 | if (id[x]) {
 | | height[tim++] = val[lst];
 | | sa[tim] = id[x];
 | | lst = x;
 | }
 | for (int c = 0; c < 26; ++c)
 | | if (son[x][c]) dfs(son[x][c]);
 | lst = fa[x];
}
int main() {
 | lst = ++cnt;
 | scanf("%s", s + 1);
 | int n = strlen(s + 1);
 | for (int i = n; i; --i) {
 | | expand(s[i] - 'a');
 | | id[lst] = i;
 | }
 | vis[1] = 1;
 | for (int i = 1; i <= cnt; ++i) if (id[i])
 | | | for (int x = i,pos = n; x && !vis[x]; x = fa[x]) {
 | | | | vis[x] = 1;
 | | | | pos -= val[x] - val[fa[x]];
 | | | | son[fa[x]][s[pos + 1] - 'a'] = x;
 | | | }
 | dfs(1);
 | for (int i = 1; i <= n; ++i) printf("%d",sa[i]); puts("");
 | for (int i = 1; i < n; ++i) printf("%d",height[i]); puts("");
 | return 0;
}
```

4.11 后缀平衡树

```cpp
const int M=1e5;
bool vis[M << 1];
char s[M];
int id[M << 1], ch[M << 1][26], height[M], tim = 0;
void dfs(int x) {
 | if (id[x]) {
 | | height[tim++] = val[lst];
 | | sa[tim] = id[x];
 | | lst = x;
 | }
 | for (int c = 0; c < 26; ++c)
 | | if (son[x][c]) dfs(son[x][c]);
 | lst = fa[x];
}
int main() {
 | lst = ++cnt;
 | scanf("%s", s + 1);
 | int n = strlen(s + 1);
 | for (int i = n; i; --i) {
 | | expand(s[i] - 'a');
 | | id[lst] = i;
 | }
 | vis[1] = 1;
 | for (int i = 1; i <= cnt; ++i) if (id[i])
 | | | for (int x = i,pos = n; x && !vis[x]; x = fa[x]) {
 | | | | vis[x] = 1;
 | | | | pos -= val[x] - val[fa[x]];
 | | | | son[fa[x]][s[pos + 1] - 'a'] = x;
 | | | }
 | dfs(1);
 | for (int i = 1; i <= n; ++i) printf("%d",sa[i]); puts("");
 | for (int i = 1; i < n; ++i) printf("%d",height[i]); puts("");
 | return 0;
}
```

Good Luck && Have Fun!