



ACM 模板

wenwenla



2017-10-22

目录

FastIO	2
计算几何基础.....	3
圆基础	6
凸包	7
并查集	8
ST 表.....	8
链表	9
树状数组.....	11
LCA	11
SCC	13
Dijkstra	14
SPFA.....	15
Dinic	17
Treap	18
线段树	21
KM 字符串匹配	22
LIS.....	23
数值积分.....	24
FWT	24
任意模数 FFT	25
SAIS.....	28

FastIO

```
namespace fastio {
    const static int buf_size = 8388608;//about 8Mb
    static char buf[buf_size];
    char *ps = buf + buf_size, *pe = buf + buf_size;
    int pos;
    bool eof = false;

    inline void read_next() {
        pe = buf + fread(buf, 1, buf_size, stdin);
        ps = buf;
        if(ps == pe) eof = true;
    }

    inline bool blank(char x) {
        return x == ' ' || x == '\n' || x == '\t' || x == '\r';
    }

    inline char nc() {
        if(ps == pe) read_next();
        return *ps++;
    }

    template<typename T>
    inline void read_num(T& res) {
        bool neg = false;
        char now = nc();
        while(blank(now)) now = nc();
        if(now == '-') {
            neg = true;
            now = nc();
        }
        T ret = 0;
        while(!blank(now)) {
            ret = ret * 10 + now - '0';
            now = nc();
        }
        res = (neg ? -ret : ret);
    }

    template<typename T>
    inline void put_num(T x) {
        if(x < 0) {
```

```

        putchar('-');
        x = -x;
    }
    if(x == 0) {
        putchar('0');
    }
    char tmp[32];
    int cnt = 0;
    while(x) {
        tmp[cnt++] = x % 10 + '0';
        x /= 10;
    }
    while(cnt > 0) {
        putchar(tmp[cnt - 1]);
        --cnt;
    }
}
}

```

计算几何基础

```

const double eps = 1e-10;
const double PI = 3.141592653589793;
double sqr(double x) { return x * x; }
int cmp(double x, double y)
{
    if(fabs(x - y) < eps) return 0;
    return (x < y) ? -1 : 1;
}

```

```

struct Point {
    double x, y;
    Point(double _x = 0, double _y = 0) : x(_x), y(_y) {}
    friend istream& operator>>(istream& is, Point& p) {
        is >> p.x >> p.y;
        return is;
    }

    bool operator<(Point rhs) const {
        return cmp(x, rhs.x) < 0 || (cmp(x, rhs.x) == 0 && cmp(y, rhs.y) < 0);
    }
    bool operator==(Point rhs) const {
        return cmp(x, rhs.x) == 0 && cmp(y, rhs.y) == 0;
    }
}

```

```

    }
    bool operator!=(Point rhs) const {
        return !(*this == rhs);
    }
};

struct Vector {
    double x, y;
    Vector(double _x = 0, double _y = 0) : x(_x), y(_y) {}
    Vector(Point s, Point t) : x(t.x - s.x), y(t.y - s.y) {}

    Vector operator+(Vector other) {
        return Vector(x + other.x, y + other.y);
    }

    Vector operator-(Vector other) {
        return Vector(x - other.x, y - other.y);
    }

    Vector operator*(double k) {
        return Vector(x * k, y * k);
    }

    double len2() const {
        return x * x + y * y;
    }

    double length() const {
        return sqrt(x * x + y * y);
    }

    Vector norm() const {
        double len = length();
        assert(len != 0);
        return Vector(x / len, y / len);
    }

    friend Vector operator*(double k, Vector other) {
        return other * k;
    }

    friend double dot(Vector lhs, Vector rhs) {
        return lhs.x * rhs.x + lhs.y * rhs.y;
    }
}

```

```

friend double cross(Vector lhs, Vector rhs) {
    return lhs.x * rhs.y - rhs.x * lhs.y;
}

friend istream& operator>>(istream& is, Vector& v) {
    cin >> v.x >> v.y;
    return is;
}
};

struct Line {
    Point o;
    Vector dir;

    Line() {}
    Line(Point x, Point y) : o(x), dir(x, y) {}
    Line(Point _o, Vector _d) : o(_o), dir(_d) {}

    friend Point getCrossPoint(Line lhs, Line rhs) {
        assert(cmp(cross(lhs.dir, rhs.dir), 0.0) != 0);
        Vector u(rhs.o, lhs.o);
        double t = cross(rhs.dir, u) / cross(lhs.dir, rhs.dir);
        Vector dt = lhs.dir * t;
        return Point(lhs.o.x + dt.x, lhs.o.y + dt.y);
    }
};

bool sameDir(Vector a, Vector b) {
    return cmp(dot(a, b), 0) > 0 && cmp(cross(a, b), 0) == 0;
}

bool parallel(Line a, Line b) {
    return cmp(cross(a.dir, b.dir), 0) == 0;
}

bool onSeg(Point a, Point b, Point c) {
    return c.x >= min(a.x, b.x) && c.y >= min(a.y, b.y) && c.x <= max(a.x, b.x) && c.y <= max(a.y,
b.y) &&
    cmp(cross(Vector(c, a), Vector(c, b)), 0) == 0;
}

double Length(Vector x) {
    return sqrt(x.x * x.x + x.y * x.y);
}

```

```

}

Vector rotate(Vector v, double cost, double sint) {
    double vx = cost * v.x - sint * v.y;
    double vy = sint * v.x + cost * v.y;
    return Vector(vx, vy);
}

Vector rotate(Vector v, double rad) {
    double c = cos(rad), s = sin(rad);
    return rotate(v, c, s);
}

double getAngle(Vector x, Vector y) {
    double dt = dot(x, y);
    return acos(dt / (Length(x) * Length(y)));
}

double toRad(double ang) {
    return ang / 180 * PI;
}

```

圆基础

```

struct Circle {
    const double PI = 3.141592653589793;
    Point c;
    double r;

    Circle(Point _c, double _r) : c(_c), r(_r) {}
    Circle(Point x, Point y, Point z) {
        Point cxy(0.5 * (x.x + y.x), 0.5 * (x.y + y.y));
        Point cyz(0.5 * (y.x + z.x), 0.5 * (y.y + z.y));
        Vector xy(x, y), yz(y, z);
        Vector nxy(-xy.y, xy.x), nyz(-yz.y, yz.x);
        c = getCrossPoint(Line(cxy, nxy), Line(cyz, nyz));
        r = Vector(c, x).length();
    }

    double area() const {
        return PI * sqr(r);
    }
}

```

```

friend void getCrossPoint(Circle a, Circle b, Point* ps, int& cnt) {
    if(a.c == b.c) return;
    int cdis2 = Vector(a.c, b.c).len2();
    int tp1 = cmp(cdis2, sqr(a.r + b.r));
    int tp2 = cmp(cdis2, sqr(a.r - b.r));
    cnt = 0;
    if(tp1 == 1 || tp2 == -1) return;
    Vector ab(a.c, b.c);
    double cost = 0.5 * (sqr(a.r) + ab.len2() - sqr(b.r)) / (ab.length() * sqr(a.r));
    double sint = sqrt(1 - cost);
    Vector v1 = rotate(ab, cost, sint).norm();
    ps[cnt++] = Point(v1.x * a.r, v1.y * a.r);
    if(tp1 == 0 || tp2 == 0) return;
    Vector v2 = rotate(ab, cost, -sint).norm();
    ps[cnt++] = Point(v2.x * a.r, v2.y * a.r);
}
};

```

凸包

```

int ConvexHull(Point* in, Point* out, int n) {
    sort(in, in + n);
    n = unique(in, in + n) - in;
    int m = 0;
    for(int i = 0; i < n; ++i) {
        while(m > 1 && cmp(cross(Vector(out[m - 2], out[m - 1]), Vector(out[m - 2], in[i])), 0) <=
0) --m;
        out[m++] = in[i];
    }
    int k = m;
    for(int i = n - 2; i >= 0; --i) {
        while(m > k && cmp(cross(Vector(out[m - 2], out[m - 1]), Vector(out[m - 2], in[i])), 0) <=
0) --m;
        out[m++] = in[i];
    }
    if(n > 1) --m;
    return m;
}

```

```

double getArea(Point* p, int n) {
    double area = 0.0;
    for(int i = 2; i < n; ++i) {
        area += cross(Vector(p[0], p[i - 1]), Vector(p[0], p[i]));
    }
}

```



```

    }
    return 0.5 * area;
}

int inPolygon(Point o, Point p[], int n) {
    int w = 0;
    for(int i = 0; i < n; ++i) {
        if(onSeg(p[i], p[(i + 1) % n], o)) return -1;//on
        int k = cmp(cross(Vector(p[i], p[(i + 1) % n]), Vector(p[i], o)), 0);
        int d1 = cmp(p[i].y - o.y, 0);
        int d2 = cmp(p[(i + 1) % n].y - o.y, 0);
        if(k > 0 && d1 <= 0 && d2 > 0) ++w;
        if(k < 0 && d2 <= 0 && d1 > 0) --w;
    }
    if(w != 0) return 1;//in
    return 0;//out
}

```

并查集

```

namespace UFS {
    const int N = 1e5 + 10;
    int fa[N];
    inline void init() { memset(fa, 0xff, sizeof(fa)); }
    inline int id(int u) {
        int rt = u;
        while(fa[rt] > 0) rt = fa[rt];
        while(fa[u] > 0) { int tmp = fa[u]; fa[u] = rt; u = tmp; }
        return rt;
    }
    inline void join(int u, int v) {
        int uf = id(u);
        int vf = id(v);
        if(uf != vf) {
            if(fa[uf] < fa[vf]) { fa[uf] += fa[vf]; fa[vf] = uf; }
            else { fa[vf] += fa[uf]; fa[uf] = vf; }
        }
    }
    inline int size(int u) { return -fa[id(u)]; }
};

```

ST 表

```

struct ST {
    //1 base, query O(1)
    int dp[50005][20];
    int n;
    int arr[50005];

    void make(int _n) {
        assert(n < 50005);
        n = _n;
        for(int i = 1; i <= n; ++i) {
            dp[i][0] = arr[i];
        }
        for(int j = 1; (1 << j) <= 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 query(int l, int r) {
        int k = 31 - __builtin_clz(r - l + 1);
        return min(dp[l][k], dp[r - (1 << k) + 1][k]);
    }
} st;

```

链表

```

template<typename T>
struct ACM_list {
    static const int MAXN = 100005;
    static const int BEG = 0;
    static const int END = MAXN - 1;
    struct Node {
        T data;
        int next;
        int pre;
    } node[MAXN];
    int free[MAXN], fp;
    int sz;

    void init() {
        node[BEG] = {T{}, END, -1};
        node[END] = {T{}, -1, BEG};
    }

```

```

        fp = MAXN - 3;
        for(int i = 0; i <= fp; ++i) {
            free[i] = i + 1;
        }
        sz = 0;
    }

    int ins_back(int idx, T val) {
        node[free[fp]].next = node[idx].next;
        node[free[fp]].data = val;
        node[free[fp]].pre = idx;
        node[idx].next = free[fp];
        node[node[free[fp]].next].pre = free[fp];
        ++sz;
        return free[fp--];
    }

    int ins_pre(int idx, T val) {
        idx = node[idx].pre;
        return ins_back(idx, val);
    }

    void del(int idx) {
        assert(idx != BEG && idx != END);
        node[node[idx].pre].next = node[idx].next;
        node[node[idx].next].pre = node[idx].pre;
        free[++fp] = idx;
        --sz;
    }

    Node& operator[](int idx) {
        return node[idx];
    }

    int begin() const {
        return node[BEG].next;
    }

    int end() const {
        return END;
    }

    int size() const {
        return sz;
    }
};

```

```
ACM_list<int> li;
```

树状数组

```
struct Bit {
    //1 Base
    int arr[300005];
    int n;

    int lowbit(int x) { return x & (-x); }

    void add(int i, int x) {
        int pos = i;
        while(pos <= n) {
            arr[pos] += x;
            pos += lowbit(pos);
        }
    }

    int sum(int i) {
        int pos = i, ans = 0;
        while(pos) {
            ans += arr[pos];
            pos -= lowbit(pos);
        }
        return ans;
    }

    void init(int _n) {
        n = _n;
        for(int i = 1; i <= n; ++i) {
            arr[i] += sum(i - 1) - sum(i - lowbit(i));
        }
    }
} bit;
```

LCA

```
const int VN = 50000, EN = 100000;
```

```
struct ST {
    //1 base, query O(1)
```

```

int dp[VN << 1][20];
int n;
int arr[VN << 1];

void make(int _n) {
    n = _n;
    for(int i = 1; i <= n; ++i) {
        dp[i][0] = i;
    }
    for(int j = 1; (1 << j) <= n; ++j) {
        for(int i = 1; (i + (1 << (j - 1))) <= n; ++i) {
            int l = dp[i][j - 1], r = dp[i + (1 << (j - 1))][j - 1];
            dp[i][j] = arr[l] < arr[r] ? l : r;
        }
    }
}

int query(int l, int r) {
    int k = 31 - __builtin_clz(r - l + 1);
    int li = dp[l][k], ri = dp[r - (1 << k) + 1][k];
    return arr[li] < arr[ri] ? li : ri;
}
} st;

struct edge {
    int to, cost, next;
    edge() {}
    edge(int _to, int _cost, int _next) : to(_to), cost(_cost), next(_next) {}
} eg[EN];
int head[VN], dis[VN], id[VN], la[VN << 1], lacnt, tot;
bool vis[VN];

void init() {
    memset(head, 0xff, sizeof(head));
    memset(vis, 0x00, sizeof(vis));
    tot = lacnt = 1;
}

void addedge(int from, int to, int cost) {
    eg[tot] = edge(to, cost, head[from]);
    head[from] = tot++;
}

void dfs(int v, int height) {

```

```

vis[v] = true;
id[v] = lacnt;
st.arr[lacnt] = height;
la[lacnt++] = v;
for(int i = head[v]; i != -1; i = eg[i].next) {
    if(!vis[eg[i].to]) {
        dis[eg[i].to] = dis[v] + eg[i].cost;
        dfs(eg[i].to, height + 1);
        st.arr[lacnt] = height;
        la[lacnt++] = v;
    }
}
}

int lca(int u, int v) {
    int l = min(id[u], id[v]), r = max(id[u], id[v]);
    return la[st.query(l, r)];
}

```

SCC

```

namespace SCC {
    //id 1 to n
    const int VN = 10005, EN = 100005;
    struct edge {
        int to, next;
        edge() {}
        edge(int _to, int _next) : to(_to), next(_next) {}
    } se[EN], re[EN];
    int shead[VN], rhead[VN], vs[VN], cmp[VN], vidx, stot, rtot, n;
    bool vis[VN];

    void init(int _n) {
        n = _n;
        stot = rtot = 0;
        memset(shead, 0xff, sizeof(shead));
        memset(rhead, 0xff, sizeof(rhead));
        vidx = 0;
    }

    void addedge(int from, int to) {
        se[stot] = edge(to, shead[from]);
        shead[from] = stot++;
    }
}

```

```

        re[rtot] = edge(from, rhead[to]);
        rhead[to] = rtot++;
    }

    void dfs(int v) {
        vis[v] = true;
        for(int i = shead[v]; i != -1; i = se[i].next) {
            if(!vis[se[i].to]) {
                dfs(se[i].to);
            }
        }
        vs[vidx++] = v;
    }

    void rdfs(int v, int k) {
        vis[v] = true;
        cmp[v] = k;
        for(int i = rhead[v]; i != -1; i = re[i].next) {
            if(!vis[re[i].to]) {
                rdfs(re[i].to, k);
            }
        }
    }

    int scc() {
        memset(vis, 0, sizeof(vis));
        for(int i = 1; i <= n; ++i) {
            if(!vis[i]) dfs(i);
        }
        memset(vis, 0, sizeof(vis));
        int k = 0;
        for(int i = vidx - 1; i >= 0; --i) {
            if(!vis[vs[i]]) rdfs(vs[i], k++);
        }
        return k;
    }
}

```

Dijkstra

```

using int64 = long long;
struct edge {
    int to, cost, next;

```

```

        edge() {}
        edge(int _to, int _cost, int _next) : to(_to), cost(_cost), next(_next) {}
    } e[12500];
    int tot;
    int head[2505];

    void init_graph() {
        tot = 0;
        memset(head, -1, sizeof(head));
    }

    void addedge(int from, int to, int cost) {
        e[tot] = edge(to, cost, head[from]);
        head[from] = tot++;
    }

    int64 dis[2505];
    bool vis[2505];

    int64 dijkstra(int s, int t) {
        using PLI = pair<int64, int>;
        memset(dis, 0x3f, sizeof(dis));
        memset(vis, 0x00, sizeof(vis));
        priority_queue<PLI, vector<PLI>, greater<PLI>> pq;
        pq.push(PLI(0, s));
        dis[s] = 0;
        while(!pq.empty()) {
            PLI now = pq.top();
            pq.pop();
            if(vis[now.second]) continue;
            vis[now.second] = true;
            for(int i = head[now.second]; i != -1; i = e[i].next) {
                if(!vis[e[i].to] && e[i].cost + now.first < dis[e[i].to]) {
                    dis[e[i].to] = e[i].cost + now.first;
                    pq.push(PLI(dis[e[i].to], e[i].to));
                }
            }
        }
        return dis[t];
    }
}

```

SPFA


```

namespace SPFA {
    const int N = 100010, M = 600010, INF = 0x3f3f3f3f;
    struct edge {
        int to, next, cost;
        edge() {}
        edge(int _to, int _next, int _cost) : to(_to), next(_next), cost(_cost) {}
    } eg[M];
    int head[N], dis[N], cnt[N], tot, incnt[N];

    void init() {
        memset(head, -1, sizeof(head));
        tot = 0;
    }

    void addedge(int from, int to, int cost) {
        eg[tot] = edge(to, head[from], cost);
        head[from] = tot++;
    }

    bool spfa(int s, int n) {
        fill(cnt, cnt + n + 1, 0);
        fill(dis, dis + n + 1, INF);
        fill(incnt, incnt + n + 1, 0);
        queue<int> q;
        q.push(s);
        dis[s] = 0; cnt[s] = 1; incnt[s] = 1;
        while(!q.empty()) {
            int now = q.front();
            q.pop();
            if(incnt[now] == 1) {
                for(int i = head[now]; i != -1; i = eg[i].next) {
                    edge& e = eg[i];
                    if(dis[e.to] > dis[now] + e.cost) {
                        dis[e.to] = dis[now] + e.cost;
                        if(++cnt[e.to] > n) return false;
                        q.push(e.to);
                        ++incnt[e.to];
                    }
                }
            }
            --incnt[now];
        }
        return true;
    }
}

```

```
}
```

Dinic

```
namespace Dinic {
    const int V = 1000010, E = 8000010, INF = 1e9;
    int vcnt;
    struct edge {
        int to, next, cap, flow;
        edge() {}
        edge(int _to, int _next, int _cap) : to(_to), next(_next), cap(_cap), flow(0) {}
    } eg[E];
    int head[V], cur[V], dis[V], que[V], qf, qe, ecnt, s, t;
    bool vis[V];

    void init(int _vcnt) {
        vcnt = _vcnt;
        memset(head, 0xff, sizeof(head[0]) * (vcnt + 1));
        ecnt = 0;
    }

    void addedge(int from, int to, int cap) {
        eg[ecnt] = edge(to, head[from], cap);
        head[from] = ecnt++;
        eg[ecnt] = edge(from, head[to], 0);
        head[to] = ecnt++;
    }

    bool bfs() {
        memset(vis, 0, sizeof(vis[0]) * (vcnt + 1));
        qf = 0; qe = 0;
        que[qe++] = s;
        dis[s] = 0; vis[s] = true;
        while(qf < qe) {
            int x = que[qf++];
            for(int i = head[x]; i != -1; i = eg[i].next) {
                const edge& e = eg[i];
                if(!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = true;
                    dis[e.to] = dis[x] + 1;
                    que[qe++] = e.to;
                    if(e.to == t) return true;
                }
            }
        }
    }
}
```

```

    }
}
return false;
}

int dfs(int x, int a) {
    if(x == t || a == 0) return a;
    int flow = 0, f;
    for(int& i = cur[x]; i != -1; i = eg[i].next) {
        edge& e = eg[i];
        if(dis[x] + 1 == dis[e.to] && (f = dfs(e.to, min(e.cap - e.flow, a)))) {
            e.flow += f;
            eg[i ^ 1].flow -= f;
            flow += f;
            a -= f;
            if(!a) break;
        }
    }
    return flow;
}

int solve(int _s, int _t) {
    s = _s; t = _t;
    int flow = 0;
    while(bfs()) {
        memcpy(cur, head, sizeof(cur[0]) * (vcnt + 1));
        flow += dfs(s, INF);
    }
    return flow;
}
}

```

Treap

```

template<typename T, class _Comp = less<T>>
struct Treap {
    const static int NODECNT = _;
    struct Node {
        int ch[2], p, sz;
        T v;
        void make(int _l, int _r, int _p, const T& _v) {
            ch[0] = _l; ch[1] = _r; p = _p; v = _v; sz = 1;
        }
    }
};

```

```

} node[NODECNT];
int m_rt, mp[NODECNT], mp_idx, node_idx;

void maintain(int x) {
    node[x].sz = 1;
    node[x].sz += node[x].ch[0] == -1 ? 0 : node[node[x].ch[0]].sz;
    node[x].sz += node[x].ch[1] == -1 ? 0 : node[node[x].ch[1]].sz;
}
_Comp cmp;

explicit Treap(const _Comp& c) : cmp(c) { unsigned seed = 19971023; srand(seed); clear(); }

Treap() : cmp(_Comp()) { unsigned seed = 19971023; srand(seed); clear(); }

void clear() { m_rt = -1; mp_idx = -1; node_idx = 0; }

void ins(const T& val) { _ins(m_rt, val); }
void _ins(int& rt, const T& val) {
    if(rt == -1) {
        if(mp_idx == -1) { node[rt = node_idx++] = make(-1, -1, rand(), val); }
        else { node[rt = mp[mp_idx--]] = make(-1, -1, rand(), val); }
    } else {
        int type = cmp(node[rt].v, val);
        _ins(node[rt].ch[type], val);
        maintain(rt);
        if(node[rt].p < node[node[rt].ch[type]].p) rotate(rt, type);
    }
}

void del(const T& val) { _del(m_rt, val); }
void _del(int& rt, const T& val) {
    assert(rt != -1);
    if(node[rt].v == val) {
        if(node[rt].ch[0] == -1) {
            mp[++mp_idx] = rt;
            rt = node[rt].ch[1];
        } else if(node[rt].ch[1] == -1) {
            mp[++mp_idx] = rt;
            rt = node[rt].ch[0];
        } else {
            int next = node[node[rt].ch[0]].p < node[node[rt].ch[1]].p;
            rotate(rt, next);
            _del(node[rt].ch[next ^ 1], val);
            maintain(rt);
        }
    }
}

```

```

    }
} else {
    _del(node[rt].ch[cmp(node[rt].v, val)], val);
    maintain(rt);
}
}

int find(const T& val) {
    int rt = m_rt;
    while(rt != -1) {
        if(node[rt].v == val) return rt;
        rt = node[rt].ch[cmp(node[rt].v, val)];
    }
    return -1;
}

void rotate(int& rt, int type) {
    int tmp = node[rt].ch[type];
    node[rt].ch[type] = node[tmp].ch[type ^ 1];
    node[tmp].ch[type ^ 1] = rt;
    maintain(rt); maintain(tmp);
    rt = tmp;
}

int kth(int k) {
    assert(k >= 1 && k <= size());
    int rt = m_rt, res = -1;
    while(rt != -1) {
        int le = node[rt].ch[0] == -1 ? 0 : node[node[rt].ch[0]].sz;
        if(le == k - 1) {
            res = node[rt].v;
            break;
        } else if(le > k - 1) {
            rt = node[rt].ch[0];
        } else {
            k -= le + 1;
            rt = node[rt].ch[1];
        }
    }
    return res;
}

int rank(const T& val) {
    int rt = m_rt, cnt = 0;

```

```

while(rt != -1) {
    int le = node[rt].ch[0] == -1 ? 0 : node[node[rt].ch[0]].sz;
    if(cmp(node[rt].v, val)) {
        cnt += le + 1;
        rt = node[rt].ch[1];
    } else {
        rt = node[rt].ch[0];
    }
}
return cnt + 1;
}

int size() { return node[m_rt].sz; }
};

```

线段树

```

using ll = long long;
struct seg_tree {
    static const int MAXN = 100005;
    struct Node {
        ll sum, tag;
    } node[MAXN << 2];
    ll arr[MAXN];

    int lson(int x) { return x << 1; }
    int rson(int x) { return (x << 1) + 1; }

    void make(int x, int xl, int xr) {
        if(xl == xr) {
            node[x].sum = arr[xl];
            node[x].tag = 0;
            return;
        }
        int mid = (xl + xr) >> 1;
        make(lson(x), xl, mid);
        make(rson(x), mid + 1, xr);
        node[x].sum = node[lson(x)].sum + node[rson(x)].sum;
        node[x].tag = 0;
    }

    void pushdown(int x, int xl, int xr) {
        node[x].sum += (xr - xl + 1) * node[x].tag;
    }
};

```

```

        if(xl < xr) {
            node[lson(x)].tag += node[x].tag;
            node[rson(x)].tag += node[x].tag;
        }
        node[x].tag = 0;
    }

    void pushup(int x, int xl, int xr) {
        int mid = (xl + xr) >> 1;
        int lcnt = mid - xl + 1, rcnt = xr - mid;
        node[x].sum = node[lson(x)].sum + lcnt * node[lson(x)].tag + node[rson(x)].sum + rcnt *
node[rson(x)].tag;
    }

    ll query(int x, int xl, int xr, int ql, int qr) {
        if(xl > qr || xr < ql) return 0;
        if(xl == ql && xr == qr) {
            return node[x].sum + (xr - xl + 1) * node[x].tag;
        }
        ll ans = 0;
        int mid = (xl + xr) >> 1;
        pushdown(x, xl, xr);
        ans += query(lson(x), xl, mid, ql, min(mid, qr));
        ans += query(rson(x), mid + 1, xr, max(ql, mid + 1), qr);
        return ans;
    }

    void update(int x, int xl, int xr, int cl, int cr, ll dt) {
        if(xr < cl || xl > cr) return;
        if(cl == xl && cr == xr) {
            node[x].tag += dt;
            return;
        }
        pushdown(x, xl, xr);
        int mid = (xl + xr) >> 1;
        update(lson(x), xl, mid, cl, min(mid, cr), dt);
        update(rson(x), mid + 1, xr, max(cl, mid + 1), cr, dt);
        pushup(x, xl, xr);
    }
} st;

```

KM 字符串匹配

```

typedef long long ll;
typedef unsigned long long ull;

int km_match(char* pattern, char* str) {
    int plen = strlen(pattern);
    int slen = strlen(str);
    int dt = max(((plen - 1) >> 1), 1);
    int cnt = 0;
    ull phash = 0, shash = 0;
    ull con = 1;
    for(int i = 0; i < plen; ++i) {
        phash = ((phash << 7) + pattern[i]);
        shash = ((shash << 7) + str[i]);
        con <<= 7;
    }
    con >>= 7;
    for(int i = 0; i < slen - plen + 1; ++i) {
        if(phash == shash) {
            bool flag = true;
            for(int j = 0; j < plen; j += dt) {
                if(pattern[j] != str[i + j]) {
                    flag = false;
                    break;
                }
            }
            if(flag) ++cnt;
        }
        shash = (((shash - str[i] * con) << 7) + str[i + plen]);
    }
    return cnt;
}

```

LIS

```

struct LIS {
    // O(nlgn), strictly increase monotonically
    const static int N = 100005;
    int a[N], b[N];

    void input(int n) {
        for(int i = 1; i <= n; ++i) {
            scanf("%d", a + i);
        }
    }
}

```



```

    }

    int solve(int n) {
        int len = 0;
        for(int i = 1; i <= n; ++i) {
            int* p = lower_bound(b + 1, b + len + 1, a[i]);
            *p = a[i];
            len = max(len, a[i] - b);
        }
        return len;
    }
} lis;

```

数值积分

```

namespace Int {
    const double eps = 1e-6;
    template<class _Callable>
    double Simpson(double lb, double ub, const _Callable& f) {
        double mid = (lb + ub) * .5;
        return (ub - lb) * (f(lb) + 4 * f(mid) + f(ub)) / 6.;
    }

    template<class _Callable>
    double _asr(double lb, double ub, double pre, double eps, const _Callable& f) {
        double mid = (lb + ub) * .5;
        double L = Simpson(lb, mid, f), R = Simpson(mid, ub, f);
        if(fabs(L + R - pre) < 15 * eps) return (L + R) + (L + R - pre) / 15.;
        return _asr(lb, mid, L, eps / 2, f) + _asr(mid, ub, R, eps / 2, f);
    }

    template<class _Callable>
    double asr(double lb, double ub, const _Callable& f) {
        return _asr(lb, ub, Simpson(lb, ub, f), eps, f);
    }
}

```

FWT

```

void FWT(int* a, int n) {
    for(int d = 1; d < n; d <= 1) {
        for(int m = d < 1, i = 0; i < n; i += m) {

```

```

        for(int j = 0; j < d; ++j) {
            int x = a[i + j], y = a[i + j + d];
            a[i + j] = (x + y) % MOD;
            a[i + j + d] = (x - y + MOD) % MOD;
            //xor:a[i+j]=x+y,a[i+j+d]=(x-y+MOD)%MOD;
            //and:a[i+j]=x+y;
            //or:a[i+j+d]=x+y;
        }
    }
}

void UFWT(int* a,int n) {
    for(int d = 1; d < n; d <= 1) {
        for(int m = d < 1, i=0; i < n; i += m){
            for(int j = 0; j < d; ++j) {
                int x = a[i + j], y = a[i + j + d];
                a[i + j] = 1LL * (x + y) * rev % MOD;
                a[i + j + d] = (1LL * (x - y) * rev % MOD + MOD) % MOD;
                //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
                //and:a[i+j]=x-y;
                //or:a[i+j+d]=y-x;
            }
        }
    }
}

```

任意模数 FFT

```

typedef long long LL;
const long double PI = acos(-1);
const int MOD = 1000000007;
const int maxn = 140100;
struct Complex {
    long double r,i;
    Complex(long double _r = 0.0, long double _i = 0.0) {
        r = _r;
        i = _i;
    }
    Complex operator +(const Complex &b) {
        return Complex(r+b.r,i+b.i);
    }
    Complex operator -(const Complex &b) {

```

```

        return Complex(r-b.r,i-b.i);
    }
    Complex operator *(const Complex &b) {
        return Complex(r*b.r-i*b.i,r*b.i+i*b.r);
    }
};

Complex conj(Complex a)
{
    return Complex(a.r,-a.i);
}

void change(Complex y[],int len)
{
    int i,j,k;
    for(i = 1, j = len/2; i < len-1; i++) {
        if(i < j)swap(y[i],y[j]);
        k = len/2;
        while( j >= k) {
            j -= k;
            k /= 2;
        }
        if(j < k) j += k;
    }
}

void FFT(Complex y[],int len,int on) //len=2^k
{
    change(y,len);
    for(int h = 2; h <= len; h <= 1) {
        Complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h));
        for(int j = 0; j < len; j+=h) {
            Complex w(1,0);
            for(int k = j; k < j+h/2; k++) {
                Complex u = y[k];
                Complex t = w*y[k+h/2];
                y[k] = u+t;
                y[k+h/2] = u-t;
                w = w*wn;
            }
        }
    }
    if(on == -1)
        for(int i = 0; i < len; i++)
            y[i].r /= len;
}

int callen(int len1,int len2)

```

```

{
    int len=1;
    while(len < (len1<<1) || len < (len2<<1))len<<=1;
    return len;
}
LL fftans[maxn];
Complex A[maxn],B[maxn],dft[4][maxn],dt[4];
int td[4];
void fft(LL* y1,int len1,LL* y2,int len2,LL mod)
{
    int len=callen(len1,len2);
    for(int x=0; x<len1; x++)A[x]=Complex(y1[x]&32767,y1[x]>>15);
    for(int x=len1; x<len; x++)A[x]=Complex(0,0);
    for(int x=0; x<len2; x++)B[x]=Complex(y2[x]&32767,y2[x]>>15);
    for(int x=len2; x<len; x++)B[x]=Complex(0,0);
    FFT(A,len,1);
    FFT(B,len,1);
    int j;
    for(int x=0; x<len; x++) {
        j=(len-x)&(len-1);
        dt[0]=(A[x]+conj(A[j]))*Complex(0.5,0);
        dt[1]=(A[x]-conj(A[j]))*Complex(0,-0.5);
        dt[2]=(B[x]+conj(B[j]))*Complex(0.5,0);
        dt[3]=(B[x]-conj(B[j]))*Complex(0,-0.5);
        dft[0][j]=dt[0]*dt[2];
        dft[1][j]=dt[0]*dt[3];
        dft[2][j]=dt[1]*dt[2];
        dft[3][j]=dt[1]*dt[3];
    }
    for(int x=0; x<len; x++) {
        A[x]=dft[0][x]+dft[1][x]*Complex(0,1);
        B[x]=dft[2][x]+dft[3][x]*Complex(0,1);
    }
    FFT(A,len,1);
    FFT(B,len,1);
    for(int x=0; x<len; x++) {
        td[0]=(LL)(A[x].r/len+0.5)%mod;
        td[1]=(LL)(A[x].i/len+0.5)%mod;
        td[2]=(LL)(B[x].r/len+0.5)%mod;
        td[3]=(LL)(B[x].i/len+0.5)%mod;
        fftans[x]=(td[0]+((LL)(td[1]+td[2])<<15)+((LL)td[3]<<30))%mod;
    }
}
LL a[maxn],b[maxn];

```

```
fft(a,n,b,n,MOD);
```

SAIS

```
template<size_t size>
struct SuffixArray {
    bool type[size<<1];
    int bucket[size],bucket1[size];
    int sa[size],rk[size],ht[size];
    inline bool isLMS(const int i,const bool *type) { return i>0&&type[i]&&!type[i-1];}
    template<class T>
    inline void inducedSort(T s,int *sa,const int len,const int cm,const int sz,bool *type,int
*bucket,int *cntbuf,int *p) {
        memset(bucket,0,sizeof(int) * cm);
        memset(sa,-1,sizeof(int) * len);
        for (int i=0;i<len;i++) bucket[s[i]]++;
        cntbuf[0]=bucket[0];
        for (int i=1;i<cm;i++) cntbuf[i]=cntbuf[i-1]+bucket[i];
        for (int i=sz-1;i>=0;i--) sa[--cntbuf[s[p[i]]]]=p[i];
        for (int i=1;i<cm;i++) cntbuf[i]=cntbuf[i-1]+bucket[i-1];
        for (int i=0;i<len;i++)
            if (sa[i]>0&&!type[sa[i]-1]) sa[cntbuf[s[sa[i]-1]]++]=sa[i]-1;
        cntbuf[0]=bucket[0];
        for (int i=1;i<cm;i++) cntbuf[i]=cntbuf[i-1]+bucket[i];
        for (int i=len-1;i>=0;i--)
            if (sa[i]>0&&type[sa[i]-1]) sa[--cntbuf[s[sa[i]-1]]]=sa[i]-1;
    }
    template<class T>
    inline void sais(T s,int *sa,int len,bool *type,int *bucket,int *bucket1,int cm) {
        int i,j,sz=0,cnt=0,p=-1,x,*cntbuf=bucket+cm;
        type[len-1]=1;
        for (i=len-2;i>=0;i--) type[i]=s[i]<s[i+1] || (s[i]==s[i+1]&&type[i+1]);
        for (i=1;i<len;i++)
            if (type[i]&&!type[i-1]) bucket1[sz++]=i;
        inducedSort(s,sa,len,cm,sz,type,bucket,cntbuf,bucket1);
        for (i=sz-1;i<len;i++)
            if (isLMS(sa[i],type)) sa[sz++]=sa[i];
        for (i=sz;i<len;i++) sa[i]=-1;
        for (i=0;i<sz;i++) {
            x=sa[i];
            for (j=0;j<len;j++) {
                if (p==-1 || s[x+j] !=s[p+j] || type[x+j] !=type[p+j]) {
                    cnt++;p=x;break;
                }
            }
        }
    }
};
```

```

        } else {
            if (j>0&&(isLMS(x+j,type)|| isLMS(p+j,type))) break;
        }
    }
    x=(~x&1 ? x>>1 : x-1>>1),sa[sz+x]=cnt-1;
}
for (i=j=len-1;i>=sz;i--)
    if (sa[i]>=0) sa[j--]=sa[i];
int *s1=sa+len-sz,*bucket2=bucket1+sz;
if (cnt<sz) {
    sais(s1,sa,sz,type+len,bucket,bucket1+sz,cnt);
} else {
    for (i=0;i<sz;i++) sa[s1[i]]=i;
}
for (i=0;i<sz;i++) bucket2[i]=bucket1[sa[i]];
inducedSort(s,sa,len,cm,sz,type,bucket,cntbuf,bucket2);
}
template<class T>
inline void getHeight(T s,int n) {
    for (int i=1;i<=n;i++) rk[sa[i]]=i;
    int j=0,k=0;
    for (int i=0;i<n;ht[rk[i++]]=k)
        for (k?k--:0,j=sa[rk[i]-1];s[i+k]==s[j+k];k++);
}
template<class T>
inline void init(T s,const int len,const int cm) {
    sais(s,sa,len,type,bucket,bucket1,cm);
    for (int i=1;i<len;i++) rk[sa[i]]=i;
    getHeight(s,len);
}
};

char s[MAXN];//Obase
SuffixArray<MAXN> sf;
sf.init(s,strlen(s)+1,256);

```