ACM-ICPC Code Library

XIA Junzhe

[update on Dec 6, 2018]

.emacs.d

```
(menu-bar-mode -1)
(tool-bar-mode -1)
(scroll-bar-mode -1)

(global-auto-revert-mode t)
(global-hl-line-mode t)
(global-linum-mode t)
(show-paren-mode t)
(desktop-save-mode t)
(delete-selection-mode t)
(add-hook 'prog-mode-hook #'hs-minor-mode)

(setq-default indent-tabs-mode nil)
(setq-default default-indent-width 4)
(setq-default c-basic-offset 4)

(setq-default compilation-read-command nil)

(set-default-font "Menlo-16")

(global-set-key (kbd "<backspace>") 'backward-delete-char)
(global-set-key (kbd "C-<backspace>") 'backward-kill-word)
(global-set-key (kbd "RET") 'newline-and-indent)
(global-set-key (kbd "C-<return>") 'compile)
(global-set-key (kbd "C-c <up>") 'windmove-up)
(global-set-key (kbd "C-c <down>") 'windmove-down)
(global-set-key (kbd "C-c <left>") 'windmove-left)
(global-set-key (kbd "C-c <right>") 'windmove-right)
(global-set-key (kbd "C-c o") 'winner-undo)
(global-set-key (kbd "C-c p") 'winner-redo)
(global-set-key (kbd "C-c i") 'hs-hide-block)
(global-set-key (kbd "C-c u") 'hs-show-block)
```

Basic

```
template<typename T>inline void chkmin(T &x, T y) { if (x >
 y) x = y; }
template<typename T>inline void chkmax(T &x, T y) { if (x <
 y) x = y; }

const long long inf = 0x3f3f3f3f3f3f3f3fll;
const int inf = 0x3f3f3f3f;

template<typename T>inline void Read(T &x)
{
    int f = 1;
    char t = getchar();
    while (t < '0' || t > '9') {
        if (t == '-') f = -1;
        t = getchar();
    }
    x = 0;
    while (t >= '0' && t <= '9') {
        x = x * 10 + t - '0';
        t = getchar();
    }
    x *= f;
}

template<typename T>inline void Write(T x)
{
    static int output[20];
    int top = 0;
    if (x < 0) putchar('-'), x = -x;
    do {
        output[++top] = x % 10;
        x /= 10;
    } while (x > 0);
    while (top > 0) putchar('0' + output[top --]);
    putchar('\n');
}
```

Data Structure

Segment Tree

```
namespace SEGT
{
    struct node
    {
        long long mx;
        long long mn;
        long long tag;
        long long cov;
        long long sum;
        long long vl;
        long long vr;
        int ln;
        int sl;
        int sm;
        int sr;

        void getcov(int l, int r, long long val)
        {
            cov = val;
            tag = 0;
            vl = vr = mx = mn = val;
            sl = sm = sr = ln = r - l + 1;
            sum = val * (r - l + 1);
        }

        void getadd(int l, int r, long long val)
        {
            if (cov < inf) cov += val;
            else tag += val;
            vl += val;
            vr += val;
            mx += val;
            mn += val;
            sum += val * (r - l + 1);
        }

        friend node operator + (const node &a, const node &
b)
        {
            node res;
            res.tag = 0, res.cov = inf;
            res.sum = a.sum + b.sum;
            res.ln = a.ln + b.ln;
            res.mx = max(a.mx, b.mx);
            res.mn = min(a.mn, b.mn);
            res.vl = a.vl;
            res.vr = b.vr;
            res.sl = a.sl;
            if (a.sm == a.ln && a.vr == b.vl) res.sl = a.sl
 + b.sl;
            res.sr = b.sr;
            if (b.sm == b.ln && b.vl == a.vr) res.sr = b.sr
 + a.sr;
            res.sm = max(a.sm, b.sm);
            if (a.vr == b.vl) chkmax(res.sm, a.sr + b.sl);
            return res;
        }
    };

    node T[maxn << 2];

    void pushdown(int cn, int l, int r)
    {
        int mid = (l + r) >> 1;
        if (T[cn].cov < inf) {
            T[cn << 1].getcov(l, mid, T[cn].cov);
            T[cn << 1 | 1].getcov(mid + 1, r, T[cn].cov);
            T[cn].cov = inf;
        }
        if (T[cn].tag) {
            T[cn << 1].getadd(l, mid, T[cn].tag);
            T[cn << 1 | 1].getadd(mid + 1, r, T[cn].tag);
            T[cn].tag = 0;
        }
    }
```

```
    void build(int cn, int l, int r)
    {
        if (l == r) {
            T[cn].getcov(l, r, a[l]);
            T[cn].cov = inf;
            return;
        }
        int mid = (l + r) >> 1;
        build(cn << 1, l, mid);
        build(cn << 1 | 1, mid + 1, r);
        T[cn] = T[cn << 1] + T[cn << 1 | 1];
    }

    void cover(int cn, int l, int r, int l0, int r0, long l
ong val)
    {
        if (l == l0 && r == r0) {
            T[cn].getcov(l, r, val);
            return;
        }
        pushdown(cn, l, r);
        int mid = (l + r) >> 1;
        if (r0 <= mid) cover(cn << 1, l, mid, l0, r0, val);
        else if (l0 > mid) cover(cn << 1 | 1, mid + 1, r, l
0, r0, val);
        else cover(cn << 1, l, mid, l0, mid, val), cover(cn
 << 1 | 1, mid + 1, r, mid + 1, r0, val);
        T[cn] = T[cn << 1] + T[cn << 1 | 1];
    }

    void add(int cn, int l, int r, int l0, int r0, long lon
g val)
    {
        if (l == l0 && r == r0) {
            T[cn].getadd(l, r, val);
            return;
        }
        pushdown(cn, l, r);
        int mid = (l + r) >> 1;
        if (r0 <= mid) add(cn << 1, l, mid, l0, r0, val);
        else if (l0 > mid) add(cn << 1 | 1, mid + 1, r, l0,
 r0, val);
        else add(cn << 1, l, mid, l0, mid, val), add(cn <<
1 | 1, mid + 1, r, mid + 1, r0, val);
        T[cn] = T[cn << 1] + T[cn << 1 | 1];
    }

    node query(int cn, int l, int r, int l0, int r0)
    {
        if (l == l0 && r == r0) return T[cn];
        node res;
        int mid = (l + r) >> 1;
        pushdown(cn, l, r);
        if (r0 <= mid) res = query(cn << 1, l, mid, l0, r0)
;
        else if (l0 > mid) res = query(cn << 1 | 1, mid + 1
, r, l0, r0);
        else res = query(cn << 1, l, mid, l0, mid) + query(
cn << 1 | 1, mid + 1, r, mid + 1, r0);
        T[cn] = T[cn << 1] + T[cn << 1 | 1];
        return res;
    }
};

/*

***Build***

SEGT::build(1, 1, n);

***Usage***

add(1, 1, n, l, r, v);
add(1, 1, n, l, r, -v);
cover(1, 1, n, l, r, v);

res = query(1, 1, n, l, r);
printf("%lld\n", res.sum); // 和
printf("%lld\n", res.mn); // 最小值
printf("%lld\n", res.mx); // 最大值
printf("%d\n", res.sm); // 最长连续序列长度
```

```
*/
```

Binary Index Tree

```
void add(int x, int v)
{
    while (x <= n) {
        c[x] += v;
        x += x & (-x);
    }
}

long long query(int x)
{
    long long res = 0;
    while (x > 0) {
        res += c[x];
        x -= x & (-x);
    }
    return res;
}
```

Splay Tree

```
namespace Splay
{
    struct node
    {
        int fa;
        int sz;
        int c[2];
        bool rev;
        bool mult;
        long long mx;
        long long mn;
        long long sum;
        long long val;
        long long tag;

        node()
        {
            fa = sz = c[0] = c[1] = 0;
            rev = mult = 0;
            mx = -inf, mn = inf;
            sum = val = tag = 0;
        }

        inline void getadd(long long v)
        {
            mx += v;
            mn += v;
            val += v;
            tag += v;
            sum += v * sz;
        }

        inline void getmult()
        {
            tag *= -1;
            sum *= -1;
            val *= -1;
            mx *= -1;
            mn *= -1;
            swap(mx, mn);
            mult ^= 1;
        }

        inline void getrev()
        {
            swap(c[0], c[1]);
            rev ^= 1;
        }
    };

#define Fa(x) T[T[x].fa]
#define Lc(x) T[T[x].c[0]]
```

```
#define Rc(x) T[T[x].c[1]]

    node T[maxn];
    int root;

    inline int newnode(long long v)
    {
        static int sz;
        sz ++;
        T[sz].val = T[sz].mx = T[sz].mn = T[sz].sum = v;
        return sz;
    }

    inline void setc(int x, int y, bool mark)
    {
        if (x) T[x].fa = y;
        if (y) T[y].c[mark] = x;
    }

    inline void update(int x)
    {
        T[x].sz = Lc(x).sz + Rc(x).sz + 1;
        T[x].mx = max(T[x].val, max(Lc(x).mx, Rc(x).mx));
        T[x].mn = min(T[x].val, min(Lc(x).mn, Rc(x).mn));
        T[x].sum = T[x].val + Lc(x).sum + Rc(x).sum;
    }

    inline void pushdown(int x)
    {
        if (T[x].mult) {
            Lc(x).getmult();
            Rc(x).getmult();
            T[x].mult = 0;
        }
        if (T[x].tag) {
            Lc(x).getadd(T[x].tag);
            Rc(x).getadd(T[x].tag);
            T[x].tag = 0;
        }
        if (T[x].rev) {
            Lc(x).getrev();
            Rc(x).getrev();
            T[x].rev = 0;
        }
    }

    void rotate(int x)
    {
        if (!T[x].fa) return;
        int p = T[x].fa;
        bool mark = (x == Fa(x).c[1]);
        setc(x, T[p].fa, p == Fa(p).c[1]);
        setc(T[x].c[mark ^ 1], p, mark);
        setc(p, x, mark ^ 1);
        update(p);
    }

    void splay(int x, int to)
    {
        static int line[maxn], top;
        int cur = x;
        line[top = 1] = cur;
        while (T[cur].fa) line[++top] = T[cur].fa, cur = T[
cur].fa;
        while (top > 0) pushdown(line[top --]);
        while (T[x].fa != to) {
            int p = T[x].fa;
            if (T[p].fa != to) {
                if ((x == Fa(x).c[1]) ^ (p == Fa(p).c[1]))
rotate(x);
                else rotate(p);
            }
            rotate(x);
        }
        update(x);
        if (to == 0) root = x;
    }

    inline int getkth(int k)
    {
        int cur = root;
        while (true) {
            pushdown(cur);
            if (Lc(cur).sz + 1 == k) break;
```

```
        else if (k <= Lc(cur).sz) cur = T[cur].c[0];
        else {
            k -= Lc(cur).sz + 1;
            cur = T[cur].c[1];
        }
    }
    return cur;
}

inline void insert(int k, long long v)
{
    int res = newnode(v);
    if (!root) {
        root = res;
    } else if (k == 0) {
        int to = root;
        pushdown(root);
        while (T[to].c[0]) {
            to = T[to].c[0];
            pushdown(to);
        }
        setc(res, to, 0);
    } else {
        int cur = getkth(k);
        if (!T[cur].c[1]) setc(res, cur, 1);
        else {
            int to = T[cur].c[1];
            pushdown(to);
            while (T[to].c[0]) {
                to = T[to].c[0];
                pushdown(to);
            }
            setc(res, to, 0);
        }
    }
    update(res);
    splay(res, 0);
}

inline void del(int x)
{
    x = getkth(x);
    splay(x, 0);
    if (!T[x].c[0] && !T[x].c[1]) {
        root = 0;
    } else if (!T[x].c[0]) {
        Rc(x).fa = 0;
        root = T[x].c[1];
    } else if (!T[x].c[1]) {
        Lc(x).fa = 0;
        root = T[x].c[0];
    } else {
        int cur = T[x].c[0];
        pushdown(cur);
        while (T[cur].c[1]) {
            cur = T[cur].c[1];
            pushdown(cur);
        }
        splay(cur, x);
        setc(T[x].c[1], cur, 1);
        update(cur);
        T[cur].fa = 0;
        root = cur;
    }
}

inline int split(int l, int r)
{
    if (l == 1 && r == T[root].sz) return root;
    else if (l == 1) {
        splay(getkth(r + 1), 0);
        return T[root].c[0];
    } else if (r == T[root].sz) {
        splay(getkth(l - 1), 0);
        return T[root].c[1];
    } else {
        splay(getkth(l - 1), 0);
        splay(getkth(r + 1), root);
        return Rc(root).c[0];
    }
}

inline long long getsum(int l, int r)
{
```

```
        int x = split(l, r);
        return T[x].sum;
    }

    inline long long getmin(int l, int r)
    {
        int x = split(l, r);
        return T[x].mn;
    }

    inline long long getmax(int l, int r)
    {
        int x = split(l, r);
        return T[x].mx;
    }

    inline void rev(int l, int r)
    {
        int x = split(l, r);
        T[x].getrev();
    }

    inline void neg(int l, int r)
    {
        int x = split(l, r);
        T[x].getmult();
        splay(x, 0);
    }

    inline void add(int l, int r, long long v)
    {
        int x = split(l, r);
        T[x].getadd(v);
        splay(x, 0);
    }
};

/*

***Usage***

insert(k, value) - insert value after the k-th value
add(l, r, z) - add z on [l, r]
rev(l, r) - reverse [l, r]
neg(l, r) - negate [l, r]
del(k) - delete the k-th value

*/
```

Union Set

```
struct dsu {
    int fa[maxn + 5];
    void init() { for (register int i = 1; i <= maxn; i++)
fa[i] = i; }
    int find(int x) { if (fa[x] == x) return x; else return
 fa[x] = find(fa[x]); }
    void merge(int x, int y) { int t1 = find(x), t2 = find(
y); if (t1 != t2) fa[t1] = t2; }
};
```

Algorithms

Edge Storage

```
int e_cn;
int fst[maxn], nxt[maxm];
int u[maxm], v[maxm], w[maxm];

void addedge(int x, int y, int z)
{
    e_cn ++;
    u[e_cn] = x, v[e_cn] = y, w[e_cn] = z;
```

```
    nxt[e_cn] = fst[x], fst[x] = e_cn;
}
```

SPFA

```
int fst[maxn], edge_cn = 0;
int u[maxn], v[maxn], w[maxn];

void spfa(int st, long long dis[maxn])
{
    static queue<int> q;
    static bool inq[maxn];
    for (int i = 1; i <= n; i++) dis[i] = inf;
    memset(inq, 0, sizeof(inq));
    q.push(st);
    dis[st] = inq[st] = 0;
    while (!q.empty()) {
        int cn = q.front();
        q.pop();
        inq[cn] = false;
        for (int i = fst[cn]; i; i = nxt[i]) {
            if (dis[v[i]] > dis[cn] + w[i]) {
                dis[v[i]] = dis[cn] + w[i];
                if (!inq[v[i]]) {
                    inq[v[i]] = true;
                    q.push(v[i]);
                }
            }
        }
    }
}

void init()
{
    memset(fst, -1, sizeof(fst));
    e_cn = -1;
}
```

Dijsktra

```
struct node
{
      int idx;
      long long dist;
      bool operator < (const node& rhs) const
      {
            return dist > rhs.dist;
      }
};

long long Dijsktra(int s, int t, int totp)
{
    static priority_queue<node> q;
    static bool vis[maxn];
    node tmp, cnt;
    while (!q.empty()) q.pop();
    memset(vis, 0, sizeof(vis));
    for (register int i = 1; i <= totp; i++) {
        dis[i] = infll;
    }
    dis[s] = 0;
    tmp.idx = s, tmp.dist = 0;
    q.push(tmp);
    while (!q.empty()) {
        cnt = q.top();
        q.pop();
        if (!vis[cnt.idx]) {
            vis[cnt.idx] = true;
            for (register int i = fst[cnt.idx]; i != -1; i
= nxt[i]) {
                if (dis[v[i]] > dis[cnt.idx] + w[i]) {
                    dis[v[i]] = dis[cnt.idx] + w[i];
                    tmp.idx = v[i];
                    tmp.dist = dis[v[i]];
                    q.push(tmp);
                }
            }
        }
```

```
    }
    return dis[t];
}
```

Dinic

```
void addone(int x, int y, int f)
{
    e_cn ++;
    v[e_cn] = y, flow[e_cn] = f;
    nxt[e_cn] = fst[x], fst[x] = e_cn;
}

void addedge(int x, int y, int f)
{
    addone(x, y, f);
    addone(y, x, 0);
}

bool label()
{
    static queue<int> q;
    memset(dis, -1, sizeof(dis));
    q.push(st);
    dis[st] = 0;
    while (!q.empty()) {
        int cn = q.front();
        q.pop();
        for (int i = fst[cn]; i != -1; i = nxt[i]) {
            if (flow[i] > 0 && dis[v[i]] == -1) {
                dis[v[i]] = dis[cn] + 1;
                q.push(v[i]);
            }
        }
    }
    return dis[ed] != -1;
}

int dinic(int cn, int now)
{
    if (now == 0 || cn == ed) return now;
    int res = 0;
    if (!cur[cn]) cur[cn] = fst[cn];
    for (int &i = cur[cn]; i != -1; i = nxt[i]) {
        if (flow[i] > 0 && dis[v[i]] == dis[cn] + 1) {
            int ret = dinic(v[i], min(now, flow[i]));
            res += ret;
            now -= ret;
            flow[i ^ 1] += ret;
            flow[i] -= ret;
            if (!now) break;
        }
    }
    if (!res) dis[cn] = -1;
    return res;
}

int maxflow()
{
    int res = 0;
    while (label()) {
        for (int i = 1; i <= sz; i++) cur[i] = 0;
        res += dinic(st, inf);
    }
    return res;
}
```

Min Cost Max Flow

```
void addedge(int x, int y, int c, int w)
{
    addone(x, y, c, w);
    addone(y, x, -c, 0);
}

bool spfa()
{
    memset(dist, inf, sizeof(dist));
    memset(vis, 0, sizeof(vis));
    q.push(source);
```

```
    dist[source] = 0, vis[source] = true;
    while (!q.empty()) {
        int cn = q.front(); q.pop();
        vis[cn] = false;
        for (register int i = fst[cn]; i != -1; i = nxt[i])
 {
            if (flow[i] > 0 && dist[v[i]] > dist[cn] + cost
[i]) {
                dist[v[i]] = dist[cn] + cost[i];
                if (!vis[v[i]]) q.push(v[i]), vis[v[i]] = t
rue;
            }
        }
    }
    return dist[sink] < inf;
}

int dfs(int cn, int cnflow)
{
    int re = 0, del;
    if (cn == sink) return cnflow;
    vis[cn] = true;
    for (register int i = fst[cn]; i != -1; i = nxt[i]) {
        if (!vis[v[i]] && flow[i] > 0 && dist[v[i]] == dist
[cn] + cost[i]) {
            del = dfs(v[i], min(cnflow, flow[i]));
            cnflow -= del;
            re += del;
            flow[i] -= del;
            flow[i ^ 1] += del;
            if (cnflow == 0) break;
        }
    }
    dist[cn] = inf, vis[cn] = false;
    return re;
}

int maxflow()
{
    int re = 0;
    while (spfa()) {
        re += dfs(source, inf) * dist[sink];
    }
    return re;
}
```

Tarjan + Rebuild

```
struct graph
{
    int fst[maxn], u[maxm], v[maxm], nxt[maxm], e_cn;
    int dfn[maxn], low[maxn], belong[maxn], deg[maxn], flag
[maxn], siz[maxn], sz, dfn_clock;
    int q[maxn], cur, r;

    bool vis[maxn];
    int endpoint, max_p, maxdis, cn_sz;

    graph()
    {
        memset(fst, -1, sizeof(fst));
        memset(flag, 0, sizeof(flag));
        memset(belong, 0, sizeof(belong));
        sz = dfn_clock = cur = 0, e_cn = -1;
    }

    inline void addedge(int x, int y)
    {
        e_cn ++;
        u[e_cn] = x, v[e_cn] = y;
        nxt[e_cn] = fst[x], fst[x] = e_cn;
    }

    void tarjan(int cn, int fa)
    {
        dfn[cn] = low[cn] = ++dfn_clock;
        q[++r] = cn, flag[cn] = 2;
        for (register int i = fst[cn]; i != -1; i = nxt[i])
 {
            if (v[i] != fa) {
                if (!flag[v[i]]) {
```

```
                    tarjan(v[i], cn);
                    low[cn] = min(low[cn], low[v[i]]);
                } else if (flag[v[i]] == 2) {
                    low[cn] = min(low[cn], dfn[v[i]]);
                }
            }
        }
        if (dfn[cn] == low[cn]) {
            cur = 0, sz ++;
            do {
                cur = q[r], r --;
                belong[cur] = sz;
                flag[cur] = 1;
            } while (cur != cn);
        }
    }

    void restruct(graph &to)
    {
        to.sz = sz;
        for (register int i = 0; i <= e_cn; i++) {
            int x = u[i], y = v[i];
            if (belong[x] != belong[y]) {
                to.addedge(belong[y], belong[x]);
                to.deg[belong[x]] ++;
            }
        }
        for (register int i = 1; i <= n; i++) to.siz[belong
[i]] ++;
    }
};
```

Cut Vertex

```
void tarjan(int cn)
{
    int edge = 0;
    dfn[cn] = low[cn] = ++dfn_clock;
    vis[cn] = true;
    for (int i = fst[cn]; i != -1; i = nxt[i]) {
        if (!dfn[v[i]]) {
            edge ++;
            tarjan(v[i]);
            chkmin(low[cn], low[v[i]]);
            if (low[v[i]] >= dfn[cn]) {
                cut[cn] = true;
            }
        } else {
            chkmin(low[cn], dfn[v[i]]);
        }
    }
    if (root == cn && edge < 2) {
        cut[cn] = false;
    }
}
```

MST

```
void build_mst()
{
    int x, y, tot = 0;
    sort(e + 1, e + en + 1);
    for (register int i = 1; i <= en; i++) {
        x = e[i].x, y = e[i].y;
        if (find(x) != find(y)) {
            merge(x, y);
            e[++tot] = e[i];
        }
    }
}
```

Link Cut Tree

```
struct Lct
{
	struct Node {
		long long sumv;
		int maxv, val, rev;
		int c[2], fa;
		Node() { maxv = -inf, sumv = val = c[0] = c[1] = fa = 0; }
	};

	Node T[maxn];
	// 0 -> left, 1 -> right

	void Set(int x, int v)
	{
		T[x].sumv = T[x].maxv = T[x].val = v;
	}

	inline bool isroot(int x)
	{
		return Fa(x).c[0] != x && Fa(x).c[1] != x;
	}

	void setc(int x, int y, bool mark)
	{
		if (y) T[y].c[mark] = x;
		if (x) T[x].fa = y;
	}

	void pushdown(int x)
	{
		if (T[x].rev) {
			T[x].rev ^= 1, Lc(x).rev ^= 1, Rc(x).rev ^= 1;
			swap(T[x].c[0], T[x].c[1]);
		}
	}

	void update(int cn)
	{
		T[cn].maxv = max(T[cn].val, max(Lc(cn).maxv, Rc(cn).maxv));
		T[cn].sumv = T[cn].val + Lc(cn).sumv + Rc(cn).sumv;
	}

	void rotate(int cn)
	{
		if (isroot(cn)) return;
		int p = T[cn].fa, mark = cn == Fa(cn).c[1];
		if (!isroot(p)) setc(cn, T[p].fa, p == Fa(p).c[1]);
		else T[cn].fa = T[p].fa;
		setc(T[cn].c[mark ^ 1], p, mark);
		setc(p, cn, mark ^ 1);
		update(p); update(cn);
	}

	void splay(int cn)
	{
		int p = T[cn].fa, r = 0;
		q[++r] = cn;
		for (register int i = cn; !isroot(i); i = T[i].fa) {
			q[++r] = T[i].fa;
		}
		while (r) pushdown(q[r--]);
		while (!isroot(cn)) {
			if (!isroot(p)) {
				if ((p == Fa(p).c[0]) ^ (cn == Fa(cn).c[0])) rotate(cn);
				else rotate(p);
			}
			rotate(cn);
			p = T[cn].fa;
		}
	}

	void access(int cn)
	{
		for (register int x = cn, r = 0; x; r = x, x = T[x].fa) {
			splay(x);
			T[x].c[1] = r;
```

```
                update(x);
            }
        }

        void makeroot(int cn)
        {
            access(cn);
            splay(cn);
            T[cn].rev ^= 1;
        }

        void link(int x, int y) // y -> x
        {
            makeroot(x);
            T[x].fa = y;
        }

        void split(int x, int y)
        {
            makeroot(x);
            access(y);
            splay(y);
        }
};
```

LCA

```
void init()
{
    dfs(1, -1);
    for (register int k = 1; (1 << k) <= n; k++) {
        for (register int i = 1; i <= n; i++) {
            if (f[i][k - 1] != -1) {
                f[i][k] = f[f[i][k - 1]][k - 1];
            }
        }
    }
}
```

```
int Lca(int x, int y)
{
    if (dep[x] < dep[y]) swap(x, y);
    for (register int i = logn; i >= 0; i--) {
        if (dep[x] - (1 << i) >= dep[y]) {
            x = f[x][i];
        }
    }
    if (x == y) return x;
    for (register int i = logn; i >= 0; i--) {
        if (f[x][i] != f[y][i]) {
            x = f[x][i];
            y = f[y][i];
        }
    }
    return f[x][0];
}
```

Heavy-Light Link Decomposition

```
void dfs1(int cn, int pr)
{
    fa[cn] = pr;
    son[cn] = -1;
    size[cn] = 1;
    for (int i = fst[cn]; i; i = nxt[i]) {
        if (v[i] != pr) {
            dfs1(v[i], cn);
            size[cn] += size[v[i]];
            if (son[cn] == -1 || size[v[i]] > size[son[cn]]
) {
                son[cn] = v[i];
            }
        }
    }
}
```

```
void dfs2(int cn, int pr)
{
    in[cn] = up[cn];
    if (son[cn] != -1) {
        up[son[cn]] = ++counter;
        tmp[counter] = val[son[cn]];
        top[son[cn]] = top[cn];
        dfs2(son[cn], cn);
    }
    for (int i = fst[cn]; i; i = nxt[i]) {
        if (v[i] != pr && v[i] != son[cn]) {
            up[v[i]] = ++counter;
            tmp[counter] = val[v[i]];
            top[v[i]] = v[i];
            dfs2(v[i], cn);
        }
    }
    out[cn] = counter;
}

int lca(int x, int y)
{
    int t1 = top[x];
    int t2 = top[y];
    while (t1 != t2) {
        if (dep[t1] < dep[t2]) {
            swap(x, y);
            swap(t1, t2);
        }
        x = fa[t1], t1 = top[x];
    }
    if (dep[x] > dep[y]) swap(x, y);
    return x;
}

int gmin(int x, int y)
{
    int res = inf;
    int t1 = top[x];
    int t2 = top[y];
    while (t1 != t2) {
        if (dep[t1] < dep[t2]) {
            swap(x, y);
            swap(t1, t2);
        }
        chkmin(res, query(1, 1, counter, up[t1], up[x]));
        x = fa[t1], t1 = top[x];
    }
    if (dep[x] > dep[y]) swap(x, y);
    chkmin(res, query(1, 1, counter, up[x], up[y]));
    return res;
}

void prepare()
{
    addedge(0, 1);
    dfs1(0, 0);
    dfs2(0, 0);
    build(1, 1, n);
}
```

SAM

```
struct node
{
    int fa, c[26];
    int val, right;
    node() {}
    node(int _val) : fa(0), val(_val), right(0) { memset(c,
 0, sizeof(c)); }
};

struct sam
{
    node T[maxn << 1];
    int sz, root, last;

    int deg[maxn << 1], right[maxn], f[maxn];
```

```
    queue<int> line;

    sam()
    {
        sz = root = last = 1;
        T[1] = node(0);
    }

    inline int newnode(int x)
    {
        T[++sz] = node(x);
        return sz;
    }

    inline void insert(int x)
    {
        int p = last, np = newnode(T[p].val + 1);
        T[np].right = 1;
        while (p && !T[p].c[x]) {
            T[p].c[x] = np;
            p = T[p].fa;
        }
        if (!p) T[np].fa = root;
        else {
            int q = T[p].c[x];
            if (T[q].val == T[p].val + 1) T[np].fa = q;
            else {
                int r = newnode(T[p].val + 1);
                memcpy(T[r].c, T[q].c, sizeof(T[r].c));
                T[r].fa = T[q].fa, T[np].fa = T[q].fa = r;
                while (p && T[p].c[x] == q) {
                    T[p].c[x] = r;
                    p = T[p].fa;
                }
            }
        }
        last = np;
    }

#define Fa(x) T[T[x].fa]

    void toposort()
    {
        memset(f, 0, sizeof(f));
        memset(deg, 0, sizeof(deg));
        for (register int i = 1; i <= sz; i++) deg[T[i].fa]
 ++;
        for (register int i = 1; i <= sz; i++) {
            if (!deg[i]) {
                line.push(i);
            }
        }
        while (!line.empty()) {
            int cn = line.front();
            line.pop();
            Fa(cn).right += T[cn].right, deg[T[cn].fa] --;
            if (!deg[T[cn].fa]) {
                line.push(T[cn].fa);
            }
        }
    }

    void solve(int n)
    {
        for (register int i = 1; i <= sz; i++) chkmax(f[T[i
].val], T[i].right);
        for (register int i = n; i >= 1; i--) chkmax(f[i -
1], f[i]);
        for (register int i = 1; i <= n; i++) printf("%d\n"
, f[i]);
    }
};

// Above : SPOJ NSUBSTR

/*

SPOJ BEADS 给一个字符串（环） 问从哪个字符开始，字典序最小。

先把原串复制一次，然后在SAM上沿着最小的边跑N步
输出right-N

SPOJ NSUBSTR 给出一个字符串，求这个字符串长度为1-n的子串的最大
```

出现次数；

用SAM上每个节点的|Right(x)|更新F(Max(x))
然后用F[i]更新F[i - 1]
之所以倒过来更新是因为每个|Right(x)|若可以包含到Max(x)，那么包含比它更短的串，是肯定没有问题的
注意SAM中的nq节点没有right值，因为它代表q和np的并

SPOJ SUBLEX 给出一个串，查询字典序排在第k个的是哪个子串

建出SAM后拓扑排序，可以按max从大到小基数排序即可
然后处理出每个点向后可以到达多少个节点
跑第k大即可

```
 void print(int cn, int k)
    {
        int to;
        while (k > 0) {
            for (register int i = 0; i < 26; i++) {
                to = T[cn].c[i];
                if (k <= T[to].sz) {
                    putchar('a' + i);
                    k --;
                    cn = to;
                    break;
                } else {
                    k -= T[to].sz;
                }
            }
        }
        putchar('\n');
    }
```

SPOJ LCS 给出两个串A,B，求A、B的最长公共子串

对A串建出后缀自动机，然后用B串在上面做匹配
实质是求出对于B串，能与A串中 *以每个r为结尾向前的最长匹配*
考虑当前状态为s，当前匹配长度为now，接下来的字符为c
如果s有c的后继，则向那个方向移动一步，now=now+1
如果没有c这个后继，只能说明当前的now不在s状态的[min(s), max(s)]范围内
考虑parent树的性质，当前s节点能匹配从s开始的一个向前的子串
那么它的祖先显然也能匹配 *这个子串的后缀*
又因为parent树中的一条向上的链所控制的范围具有连续性（max[fa] = min[son] - 1）
因此一直向上找，第一个出现x这个后继的状态的max就是我们要求的，以当前r结尾向前的最长匹配
不停取max即可。

```
int go(char *str, int n)
    {
        int cn = root, res = 0, now = 0, ch;
        for (register int i = 1; i <= n; i++) {
            ch = str[i] - 'a';
            if (T[cn].c[ch]) {
                now ++;
                cn = T[cn].c[ch];
            } else {
                while (cn && !T[cn].c[ch]) cn = T[cn].fa;
                if (!cn) cn = root, now = 0;
                else now = T[cn].val + 1, cn = T[cn].c[ch];
            }
            res = max(res, now);
        }
        return res;
    }

*/
```

AC Automaton

```
namespace ACautomaton
{
    struct node
    {
        int c[26];
        int fail;
        node() { memset(c, 0, sizeof(c)), fail = 0; }
```

```
};

int sz = 1;
node T[maxnode];
bool match[maxnode];

void insert(char *str)
{
    int cur = 1;
    for (int i = 1; str[i]; i++) {
        int v = str[i] - 'a';
        if (!T[cur].c[v]) T[cur].c[v] = ++sz;
        cur = T[cur].c[v];
    }
    match[cur] = true;
}

void build()
{
    static queue<int> q;
    for (int i = 0; i < 26; i++) T[0].c[i] = 1;
    q.push(1);
    while (!q.empty()) {
        int cn = q.front();
        q.pop();
        for (int i = 0; i < 26; i++) {
            if (!T[cn].c[i]) {
                T[cn].c[i] = T[T[cn].fail].c[i];
                continue;
            }
            int t = T[cn].fail;
            while (!T[t].c[i]) t = T[t].fail;
            T[T[cn].c[i]].fail = T[t].c[i];
            q.push(T[cn].c[i]);
            match[cn] |= match[T[cn].fail];
        }
    }
}

int go(int cn, int c)
{
    while (!T[cn].c[c]) cn = T[cn].fail;
        return T[cn].c[c];
    }
};
```

Computational Geometry Basic

```
struct point {
    double x, y;
    void read() { cin >> x >> y; }
    point() {}
    point(double x, double y) : x(x), y(y) {}
    double len() { return sqrt(x * x + y * y); }
};

struct circle {
    point c;
    double r;
    circle() {}
    circle(point c, double r) : c(c), r(r) {}
};

struct line
{
    point c;
    point v;
    double angle;

    line() {}
    line(point c, point v) : c(c), v(v) { angle = atan2(
v.y, v.x); }
};

int dcmp(double x) { if (fabs(x) < eps) return 0; else r
eturn x > 0 ? 1 : -1; }

bool operator == (const point &a, const point &b) { retu
```

```
rn dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0; }
bool operator != (const point &a, const point &b) { retu
rn dcmp(a.x - b.x) || dcmp(a.y - b.y); }
bool operator < (const point &a, const point &b) { retur
n a.x < b.x || (a.x == b.x && a.y < b.y); }
point operator + (const point &a, const point &b) { retu
rn point(a.x + b.x, a.y + b.y); }
point operator - (const point &a, const point &b) { retu
rn point(a.x - b.x, a.y - b.y); }
point operator * (const point &a, double v) { return poi
nt(a.x * v, a.y * v); }
point operator / (const point &a, double v) { return poi
nt(a.x / v, a.y / v); }
double operator ^ (const point a, const point b) { retur
n a.x * b.y - a.y * b.x; }
double operator * (const point a, const point b) { retur
n a.x * b.x + a.y * b.y; }
double dis (const point &a, const point &b) { return sqr
t((b.x - a.x) * (b.x - a.x) + (b.y - a.y) * (b.y - a.y))
; }
double getang (const point &v1, const point &v2) { retur
n acos((v1 * v2) / v1.len() / v2.len()); }
```

Intersection of Segments

```
point intersection(const point& a, const point& b, const
 point& c, const point& d)
{
      double u = ((a - c) ^ (d - c)) / ((d - c) ^ (b - a
));
      return a + (b - a) * u;
}

point intersection(const line& a, const line &b)
{
      double u = ((a.c - b.c) ^ b.v) / (b.v ^ a.v);
    return a.c + a.v * u;
}
```

Is segment AB and CD intersected?

```
bool intersect(const point &a, const point &b, const poi
nt &c, const point &d)
{
    return ((b - a) ^ (c - a)) * ((b - a) ^ (d - a)) < 0
 && ((d - c) ^ (a - c)) * ((d - c) ^ (b - c)) < 0;
}
```

Rotate a vector

```
point rotate(point v, double rad)
{
    double cosx = cos(rad);
    double sinx = sin(rad);
    return point(v.x * cosx - v.y * sinx, v.x * sinx + v
.y * cosx);
}
```

Is P inside a polygon?

```
bool inside(const point &x, vector<int> &Cycle)
{
      int pri1 = 998244353;
```

```
    int pri2 = 1e9 + 7;
    int cur = 0;
    point y = point(pri1, pri2);
    for (int i = 0; i < Cycle.size(); i++) {
            cur += intersect(x, y, g[e[Cycle[i]].u], g[e
[Cycle[i]].v]);
    }
    return cur & 1;
}
```

Intersection Points of Two Circle

```
bool inter(const circle &a, const circle &b, pair<point,
 point> &re)
{
    double d = (b.c - a.c).len();
    if (dcmp(d - a.r - b.r) >= 0) return false;
    double cs = (sqr(a.r) + sqr(d) - sqr(b.r)) / (2 * a.
r * d);
    double sn = sqrt(1 - sqr(cs));
    point dir = (b.c - a.c) / d * a.r;
    point d1 = rotate(dir, cs, sn);
    point d2 = rotate(dir, cs, -sn);
    re = make_pair(a.c + d1, a.c + d2);
    return true;
}
```

Intersection Region of Multiple Circles (even union set)

[reference: https://blog.csdn.net/wty_/article/details/12835403]

```
struct cp
{
    double x, y, r, angle;
    int d;
    cp() {}
    cp(double xx, double yy, double ang = 0, int t = 0)
    {
        x = xx;
        y = yy;
        angle = ang;
        d = t;
    }
    void get()
    {
        scanf("%lf%lf%lf", &x, &y, &r);
        d = 1;
    }
} cir[maxn], tp[maxn << 1];

double dis(cp a, cp b)
{
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}

double cross(cp p0, cp p1, cp p2)
{
    return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y)
 * (p2.x - p0.x);
}

int CirCrossCir(cp p1, double r1, cp p2, double r2, cp &
cp1, cp &cp2)
{
    double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx
* mx;
    double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my
* my;
    double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (s
q - sqr(r1 + r2));
    if (d + eps < 0) return 0;
```

```
    if (d < eps) d = 0;
    else d = sqrt(d);
    double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) +
sx * my2;
    double y = my * ((r1 + r2) * (r1 - r2) + my * sy) +
sy * mx2;
    double dx = mx * d, dy = my * d;
    sq *= 2;
    cp1.x = (x - dy) / sq;
    cp1.y = (y + dx) / sq;
    cp2.x = (x + dy) / sq;
    cp2.y = (y - dx) / sq;
    if (d > eps) return 2;
    else return 1;
}

bool circmp(const cp& u, const cp& v)
{
    return dcmp(u.r - v.r) < 0;
}

bool cmp(const cp& u, const cp& v)
{
    if (dcmp(u.angle - v.angle)) return u.angle < v.angl
e;
    return u.d > v.d;
}



double calc(cp cir, cp cp1, cp cp2)
{
    double ans = (cp2.angle - cp1.angle) * sqr(cir.r)
                 - cross(cir, cp1, cp2) + cross(cp(0, 0)
, cp1, cp2);
    return ans / 2;
}

void CirUnion(cp cir[], int n)
{
    cp cp1, cp2;
    sort(cir, cir + n, circmp);
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            if (dcmp(dis(cir[i], cir[j]) + cir[i].r - ci
r[j].r) <= 0)
                cir[i].d++;
    for (int i = 0; i < n; ++i)
    {
        int tn = 0, cnt = 0;
        for (int j = 0; j < n; ++j)
        {
            if (i == j) continue;
            if (CirCrossCir(cir[i], cir[i].r, cir[j], ci
r[j].r,
                            cp2, cp1) < 2) continue;
            cp1.angle = atan2(cp1.y - cir[i].y, cp1.x -
cir[i].x);
            cp2.angle = atan2(cp2.y - cir[i].y, cp2.x -
cir[i].x);
            cp1.d = 1;
            tp[tn++] = cp1;
            cp2.d = -1;
            tp[tn++] = cp2;
            if (dcmp(cp1.angle - cp2.angle) > 0) cnt++;
        }
        tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, PI,
 -cnt);
        tp[tn++] = cp(cir[i].x - cir[i].r, cir[i].y, -PI
, cnt);
        sort(tp, tp + tn, cmp);
        int p, s = cir[i].d + tp[0].d;
        for (int j = 1; j < tn; ++j)
        {
            p = s;
            s += tp[j].d;
            area[p] += calc(cir[i], tp[j - 1], tp[j]);
        }
    }
}
```

```
double area()
{
    CirUnion(cir, n);
    for (int i = 1; i <= n; ++i) {
        area[i] -= area[i + 1];
    }
    //area[i]为重叠了i 次的面积
    //tot 为总面积
    double tot = 0;
    for(int i = 1; i <= n; i++) tot += area[i];
      return tot;
}
```

(Directional) Area of Intersection Region Between Circle and Triangle (Share the Center)

```
double _sec(const circle &C, const point &A, const point
 &B)
{
    // area of a sector
    point v1 = A - C;
    point v2 = B - C;
    double ang = acos((v1 * v2) / v1.len() / v2.len());
    return abs(0.5 * r * r * ang);
}

double intersection(const circle &C, const point &s, con
st point &t)
{
    // the triangle is (C.c, s, t)
    double ret = 0;
    point v1 = s - C.c;
    point v2 = t - C.c;
    bool neg = ((v1 ^ v2) < 0);
    vector<point> int1 = line_intersection(C, s, t);
    vector<point> int2 = line_intersection(C, C.c, t);
    vector<point> int3 = line_intersection(C, C.c, s);
    if (dcmp(v1 ^ v2) == 0) {
        ret = 0;
    } else if ((int2.size() == 0 && int3.size() == 0)
               || (int2.size() == 0 && int1.size() == 0)
               || (int3.size() == 0 && int1.size() == 0)
) {
        ret = abs(0.5 * (v1 ^ v2));
    } else if (int3.size() == 0) {
        point A = int1[0];
        point B = int2[0];
        ret = abs(0.5 * (v1 ^ (A - C.c))) + _sec(C, A, B
);
    } else if (int2.size() == 0) {
        point A = int3[0];
        point B = int1[0];
        ret += _sec(C, A, B) + abs(0.5 * ((B - C.c) ^ v2
));
    } else {
        vector<point> inters = line_intersection(C, s, t
);
        if (inters.size() < 2) {
            ret = _sec(C, s, t);
        } else {
            point A = inters[0];
            point B = inters[1];
            point P = int3[0];
            point Q = int2[0];
            ret = abs(0.5 * ((A - C) ^ (B - C)));
            ret += min(_sec(C, P, A), _sec(C, P, B));
            ret += min(_sec(C, Q, A), _sec(C, Q, B));
        }
        if (neg) ret *= -1;
    }
    return ret;
}
```

Symmetric Point

```
point symmetry(point p, point t1, point t2)
{
    point mid = t2 - t1;
    point side = p - t1;
    double ang = getang(mid, side);
    bool flag = (dcmp(side ^ mid) > 0);
    point res = flag ? rotate(side, ang * 2) : rotate(si
de, -ang * 2);
    return res + t1;
}
```

Half-Plane Intersection

```
bool operator < (const line &a, const line &b)
{
    if (a.angle == b.angle) return dcmp((b.c - a.c) ^ a.
v) > 0;
    else return a.angle < b.angle;
}

bool onleft(const line &a, const line &b, const line &c)
{
    point res = intersection(a, b);
    return ((res - c.c) ^ c.v) <= 0;
}

void init(line arr[maxn], int &len)
{
    arr[++len] = line(point(-inf, -inf), point(inf, 0));
    arr[++len] = line(point(-inf, inf), point(0, -inf));
    arr[++len] = line(point(inf, inf), point(-inf, 0));
    arr[++len] = line(point(inf, -inf), point(0, inf));
}

vector<point> half_plane_intersection()
{
    static line seg[maxn];
    static line q[maxn];
      vector<point> res;
    int h = 0, r = -1, cnt = 0;
    sort(seg + 1, seg + n + 1);
    for (int i = 1; i <= n; i++) {
            if (i == 1 || seg[i].angle != seg[i - 1].ang
le) seg[++cnt] = seg[i];
    }
    q[++r] = seg[1];
    q[++r] = seg[2];
    for (int i = 3; i <= cnt; i++) {
        while (r > h && !onleft(q[r], q[r - 1], seg[i]))
 r --;
        while (r > h && !onleft(q[h], q[h + 1], seg[i]))
 h ++;
        q[++r] = seg[i];
    }
    while (r > h && !onleft(q[r], q[r - 1], q[h])) r --;
    while (r > h && !onleft(q[h], q[h + 1], q[r])) h ++;
    q[r + 1] = q[h];
    for (int i = h; i <= r; i++) {
            res.push_back(intersection(q[i], q[i + 1]));
    }
    return res;
}
```

Convex Hull

```
void init(point pt[maxn])
{
    static pair<pair<double, double>, point> foo [,maxn]
;
```

```
    int minpos = 1;
    for (int i = 1; i <= n; i++) {
        if (pt[i] < pt[minpos]) {
            minpos = i;
        }
    }
    swap(pt[1], pt[minpos]);
    for (int i = 2; i <= n; i++) {
        foo[i - 1] = make_pair(make_pair(atan2(pt[i].y -
 pt[1].y, pt[i].x - pt[1].x),
                                          dis(pt[i], pt[1
])), pt[i]);
    }
    sort(foo + 1, foo + n);
    for (int i = 1; i < n; i++) {
        pt[i + 1] = foo[i].second;
    }
}

void convex_hull()
{
    conv_n = 0;
    for (int i = 1; i <= n; i++) {
        while (conv_n > 1 && dcmp((pt[i] - conv[conv_n -
 1]) ^ (conv[conv_n] - conv[conv_n - 1])) >= 0) {
            conv_n --;
        }
        conv[++conv_n] = pt[i];
    }
    for (int i = 0; i < conv_n; i++) {
        conv[i] = conv[i + 1];
    }
}
```