近期模板

KMP

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
#include <cstdio>
using namespace std;
const int MAX = 1e6 + 10;
char pat[MAX];
char tar[MAX];
int nt[MAX], ans;

void getNext()
{
    int lenp = strlen(pat);
    int lent = strlen(tar);
    memset(nt, 0, sizeof nt);
    nt[0] = -1;
    int i = 0;
    int j = -1;
    while (i < lenp) {
        if (j == -1 || pat[i] == pat[j]) {
            nt[ ++ i] = ++ j;
        } else {
            j = nt[j];
        }
    }
}

void kmp()
{
    ans = 0;
    int i = 0, j = 0;
    int lent = strlen(tar);
    int lenp = strlen(pat);
    while (i < lent) {
        if (tar[i] == pat[j] || j == -1) {
            i ++; j ++;
        } else {
            j = nt[j];
        }
        if (j == lenp) {
            ans ++;
            j = nt[j];
        }
    }
}

int main()
{
    int t;
    scanf("%d", &t);
    while (t --) {
        scanf(" %s %s", pat, tar);
        getNext();
        kmp();
        printf("%d\n", ans);
    }
}
```

LCIS 最大公共上升子序列

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 510;
int n, m, a[MAX], b[MAX];
// dp[i][j] 表示a数组前i个，b数组前j个，且b[j]在公共上升序列中的最大长度
// 压成一维，LIS和LCS的思想合起来
int dp[MAX];

int main()
{
    int tc;
    scanf("%d", &tc);
    while (tc --) {
        memset(dp, 0, sizeof dp);
        scanf("%d", &n);
        for (int i = 1; i <= n; i ++) {
            scanf("%d", &a[i]);
        }
        scanf("%d", &m);
        for (int i = 1; i <= m; i ++) {
            scanf("%d", &b[i]);
        }
        int ans = -1;
        for (int i = 1; i <= n; i ++) {
            int maxx = 0;
            for (int j = 1; j <= m; j ++) {
                if (a[i] == b[j]) {
                    dp[j] = max(dp[j], maxx + 1);
                }
                if (b[j] < a[i]) {
                    maxx = max(maxx, dp[j]);
                }
                ans = max(ans, dp[j]);
            }
        }
        printf("%d\n", ans);
        if (tc != 0)
            puts("");
    }
    return 0;
}
```

AC自动机静态模板

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e6 + 10;
int n, ans;

struct acm {
    static int tot;
    int trie[MAX][26], sum[MAX], fail[MAX];
    void init() {
        memset(sum, 0, sizeof(int) * (tot + 1));
        memset(fail, 0, sizeof(int) * (tot + 1));
        for (int i = 0; i <= tot; i ++) {
            for (int j = 0; j < 26; j ++) {
                trie[i][j] = 0;
            }
        }
        fail[0] = 0;
        tot = 0;
    }
    void add(char *s, int len) {
        int x = 0;
        for (int i = 0; i < len; i ++) {
            int id = s[i] - 'a';
            if (!trie[x][id]) {
                trie[x][id] = ++ tot;
            }
            x = trie[x][id];
            if (i == len - 1) {
                sum[x] ++;
            }
        }
    }
    int getfail(int x, int k) {
        if (trie[x][k]) return trie[x][k];
        if (x == 0) return 0;
        return getfail(fail[x], k);
    }
    void makefail() {
        queue<int> q;
        q.push(0);
        while (!q.empty()) {
            int now = q.front(); q.pop();
            for (int i = 0; i < 26; i ++) {
                if (trie[now][i]) {
                    if (now == 0) {
                        fail[trie[now][i]] = 0;
                    } else {
                        fail[trie[now][i]] = getfail(fail[now], i);
                    }
                    q.push(trie[now][i]);
                }
            }
        }
    }
    void match(char *s, int len) {
        int x = 0;
        for (int i = 0; i < len; i ++) {
            int id = s[i] - 'a';
            while (x && !trie[x][id]) x = fail[x];
            x = trie[x][id];
            int temp = x;
            while (temp) {
                if (sum[temp]) {
                    ans += sum[temp];
                    sum[temp] = 0;
                }
                temp = fail[temp];
            }
        }
    }
} ac;
int acm::tot = MAX;
int main()
{
    int t;
    scanf("%d", &t);
    while (t --) {
        ans = 0;
        char s[MAX];
        ac.init();
        scanf("%d", &n);
        for (int i = 1; i <= n; i ++) {
            scanf(" %s", s);
            ac.add(s, strlen(s));
        }
        ac.makefail();
        scanf(" %s", s);
        ac.match(s, strlen(s));
        printf("%d\n", ans);
    }
    return 0;
}
```

LCA Tarjan

```
// tarjan lca模板
/**
* 核心思想：如果要求u，v的lca，有一个点a，
* uv分别在a的左右子树，那么a就是uv的lca
* 用到了并查集
*/
#include <bits/stdc++.h>
using namespace std;
const int MAX = 4e4 + 10;
struct edge {
    int v, w, nt;
} e[MAX << 1];
// 离线算法，需要存储所有查询
struct Query {
    int id, v;
    Query(int vv, int idd): id(idd), v(vv) {}
};
int head[MAX], father[MAX], vis[MAX];
// 用来存储询问的节点和答案
int lca[MAX], ulca[MAX], vlca[MAX];
long long dis[MAX];
int cnte, n, m;
// 存储所有询问
vector<Query> query[MAX];

void init()
{
    cnte = 0;
    memset(head, 0, sizeof head);
    memset(vis, 0, sizeof vis);
    for (int i = 1; i <= n; i ++) {
        father[i] = i;
    }
}

int find(int x)
{
    while (x != father[x]) {
        father[x] = father[father[x]];
        x = father[x];
    }
    return x;
}

// 并查集合并，注意tarjan求lca的时候，需要把孩子节点
// 并到父亲节点上，因此合并的时候谁并到谁的次序不能错！
// 把y并到x
void merge(int x, int y)
{
    x = find(x);
    y = find(y);
    father[y] = x;
}

void add(int u, int v, int w)
{
    ++ cnte;
    e[cnte].v = v;
    e[cnte].w = w;
    e[cnte].nt = head[u];
    head[u] = cnte;
}

// 核心部分
void tarjan(int u, int fa)
{
    // dfs
    for (int i = head[u]; i; i = e[i].nt) {
        int v = e[i].v;
        if (v != fa) {
            dis[v] = dis[u] + e[i].w;
            tarjan(v, u);
        }
    }
    // 可以把这些查询看作图的深度优先搜索树上的非树边
    for (int i = 0; i < query[u].size(); i ++) {
        int v = query[u][i].v;
        int id = query[u][i].id;
        // 如果v被访问过了，这条边如果相当于前向边（祖先指向孩子）
        // 此时find(v) = u
        // 或者如果是交叉边（无祖先关系）
        // 此时find(v)是使uv在其不同子树的节点
        if (vis[v]) {
            lca[id] = find(v);
        }
    }
    // 当该节点及其子树所有节点都访问过，才把当前节点并到父节点上
    // 并且这时才置访问标记
    merge(fa, u);
    // 第一次访问到该节点就置访问标记也对，
    // 不过会使uv有直接祖先关系的查询查询两次
    vis[u] = 1;
}

int main()
{
    int t;
    scanf("%d", &t);
    while (t --) {
        scanf("%d%d", &n, &m);
        init();
        for (int i = 1; i < n; i ++) {
            int u, v, w;
            scanf("%d%d%d", &u, &v, &w);
            add(u, v, w);
            add(v, u, w);
        }
        for (int i = 1; i <= m; i ++) {
            int u, v;
            scanf("%d%d", &u, &v);
            ulca[i] = u;
            vlca[i] = v;
            query[u].push_back(Query(v, i));
            query[v].push_back(Query(u, i));
        }
        dis[1] = 0;
        tarjan(1, 0);
        for (int i = 1; i <= m; i ++) {
            printf("%lld\n", dis[ulca[i]] + dis[vlca[i]] - 2 * dis[lca[i]]);
        }
    }
    return 0;
}
```

LCA 倍增

```
// 倍增求LCA
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>
using namespace std;
const int MAX = 1e4 + 10;
// 最大深度对二取对数
const int MAXH = 16;
queue<int> q;
struct edge {
    int v, nt;
} e[MAX];
int n, isroot[MAX], head[MAX], cnte, dep[MAX];
// anc[i][j]表示第i个节点向上跳2^j层后的节点
// anc[i][0]就是父亲节点
// 如果跳2^j层后超过了root，那么anc[i][j] = root
int anc[MAX][MAXH];

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

void add(int u, int v)
```

```cpp
{
    ++ cnte;
    e[cnte].v = v;
    e[cnte].nt = head[u];
    head[u] = cnte;
}

// x向上跳h层后的节点编号
int swim(int x, int h)
{
    int ret = x;
    // 从二进制角度看，如6=110，那么先跳2层，再跳4层
    for (int i = 0; h; i ++, h >>= 1) {
        if (h & 1) {
            ret = anc[ret][i];
        }
    }
    return ret;
}

// 遍历整个树，打出anc表
void bfs(int root)
{
    dep[root] = 1;
    q.push(root);
    for (int i = 0; i < MAXH; i ++) {
        anc[root][i] = root;
    }
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int i = head[u]; i; i = e[i].nt) {
            int v = e[i].v;
            if (v != anc[u][0]) {
                dep[v] = dep[u] + 1;
                anc[v][0] = u;
                for (int i = 1; i < MAXH; i ++) {
                    // 倍增
                    anc[v][i] = anc[anc[v][i - 1]][i - 1];
                }
                q.push(v);
            }
        }
    }
}

int lca(int x, int y)
{
    if (dep[x] < dep[y]) {
        swap(x, y);
    }
    // 先把较深的跳到较浅的同一高度
    x = swim(x, dep[x] - dep[y]);
    if (x == y) {
        return x;
    }
    // 后一次跳的高度一定比前一次跳的高度还小
    // 可以用反证法证明
    for (int i = MAXH - 1; i >= 0; i --) {
        if (anc[x][i] != anc[y][i]) {
            x = anc[x][i];
            y = anc[y][i];
        }
    }
    // 循环结束后，anc[x][0] = anc[y][0] = lca
    return anc[x][0];
}

int main()
{
    int t;
    scanf("%d", &t);
    while (t --) {
        init();
        scanf("%d", &n);
        int u, v;
        for (int i = 1; i < n; i ++) {
            scanf("%d%d", &u, &v);
            add(u, v);
            isroot[v] = 0;
        }
        int root = -1;
        for (int i = 1; i <= n; i ++) {
            if (isroot[i]) {
                root = i;
                break;
            }
        }
        bfs(root);
        scanf("%d%d", &u, &v);
        printf("%d\n", lca(u, v));
    }
    return 0;
}
```

强连通分量 Tarjan

```
#include <cstdio>
#include <cstring>
#include <stack>
using namespace std;
const int MAXN = 110;
const int MAXM = MAXN * MAXN;
struct edge {
    int v, nt;
```

```
} e[MAXM];
int head[MAXN], low[MAXN], dfn[MAXN], in[MAXN], out[MAXN], color[MAXN];
int cnte, cntc, idx, n;
stack<int> s;

void init()
{
    cnte = cntc = idx = 0;
    memset(head, 0, sizeof head);
    memset(dfn, 0, sizeof dfn);
    memset(color, 0, sizeof color);
    memset(in, 0, sizeof in);
    memset(out, 0, sizeof out);
}

void add (int u, int v)
{
    cnte ++;
    e[cnte].v = v;
    e[cnte].nt = head[u];
    head[u] = cnte;
}

void tarjan(int u)
{
    dfn[u] = low[u] = ++ idx;
    s.push(u);
    for (int i = head[u]; i; i = e[i].nt) {
        int v = e[i].v;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (!color[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        ++ cntc;
        while (true) {
            int now = s.top(); s.pop();
            color[now] = cntc;
            if (now == u) {
                break;
            }
        }
    }
}

int main()
{
    while (~scanf("%d", &n)) {
        init();
        for (int i = 1; i <= n; i ++) {
            int v;
            while (scanf("%d", &v), v) {
                add(i, v);
            }
        }
        for (int i = 1; i <= n; i ++) {
            if (!dfn[i]) {
                tarjan(i);
            }
        }

        for (int u = 1; u <= n; u ++) {
            for (int i = head[u]; i; i = e[i].nt) {
                int v = e[i].v;
                if (color[u] != color[v]) {
                    in[color[v]] ++;
                    out[color[u]] ++;
                }
            }
        }
        int in0 = 0, out0 = 0;
        for (int i = 1; i <= cntc; i ++) {
            if (in[i] == 0) ++ in0;
            if (out[i] == 0) ++ out0;
        }
        printf("%d\n", in0);
        if (cntc == 1) {
            puts("0");
        } else {
            printf("%d\n", max(in0, out0));
        }

    }
    return 0;
}
```

双联同分量 桥

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <stack>
#include <queue>
using namespace std;
const int MAX = 100010;

struct edge {
    int v, nt, used;
} e[MAX << 2];
int n, m, q, cnte, idx, cntb;
int head[MAX], dfn[MAX], low[MAX];
int isbridge[MAX], father[MAX];

void init()
{
    father[1] = cnte = idx = cntb = 0;
    memset(head, -1, sizeof head);
    memset(dfn, 0, sizeof dfn);
    memset(isbridge, 0, sizeof isbridge);
}

void add(int u, int v)
{
    e[cnte].v = v;
    e[cnte].nt = head[u];
    e[cnte].used = 0;
    head[u] = cnte;
    ++ cnte;
}

void tarjan(int u)
{
    dfn[u] = low[u] = ++ idx;
    for (int i = head[u]; i != -1; i = e[i].nt) {
        if (e[i].used)
            continue;
        e[i].used = 1;
        e[i ^ 1].used = 1;
        int v = e[i].v;
        if (!dfn[v]) {
            father[v] = u;
            tarjan(v);
            if (low[v] > dfn[u]) {
                isbridge[v] = 1;
                ++ cntb;
            }
            low[u] = min(low[u], low[v]);
        } else {
            low[u] = min(low[u], dfn[v]);
        }
    }
}

void lca(int u, int v)
{
    while (u != v) {
        while (dfn[u] > dfn[v]) {
            if (isbridge[u]) {
                isbridge[u] = 0;
                -- cntb;
            }
            u = father[u];
        }
        while (dfn[v] > dfn[u]) {
            if (isbridge[v]) {
                isbridge[v] = 0;
                -- cntb;
            }
            v = father[v];
        }
    }
    //printf("lca : %d\n", u);
}

int main()
{
    int tc = 1;
    while (~scanf("%d%d", &n, &m), n || m) {
        init();
        int u, v;
        for (int i = 1; i <= m; i ++) {
            scanf("%d%d", &u, &v);
            add(u, v);
            add(v, u);
        }

        tarjan(1);

        scanf("%d", &q);
        printf("Case %d:\n", tc ++);
        while (q --) {
            int u, v;
            scanf("%d%d", &u, &v);
            lca(u, v);
            printf("%d\n", cntb);
        }
        puts("");
    }

    return 0;
}
```