省选模板

panda_2134

2018 年 4 月 1 日

目录

1 图论

1.1 强连通分量

1.2 点双连通分量

1.3 边双连通分量

2 网络流

2.1 最大流

MaximumFlow.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Edge {
    int v, flow, cap, next;
};

const int MAXN = 1e4, MAXM = 1e5, INF = 0x3f3f3f3f;
int n, m, s, t, e_ptr = 1, head[MAXN+10]; Edge E[(MAXM+10)<<1];
int d[MAXN+10], cur[MAXN+10];

void AddEdge(int u, int v, int cap) {
    E[++e_ptr] = (Edge) { v, 0, cap, head[u] }; head[u] = e_ptr;
    E[++e_ptr] = (Edge) { u, 0,   0, head[v] }; head[v] = e_ptr;
}

bool BFS() {
    queue<int> Q;
    memset(d, 0xff, sizeof(d));
    Q.push(s); d[s] = 0;
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        for(int j=head[u]; j; j=E[j].next) {
            int v = E[j].v, f = E[j].flow, c = E[j].cap;
            if(f < c && d[v] == -1) {
                d[v] = d[u] + 1;
                if(v == t) return true;
                else Q.push(v);
            }
        }
    }
    return false;
}

int DFS(int u, int flow) {
    if(u == t || flow == 0) return flow; // !!!!!
    int res = flow;
    for(int &j=cur[u]; j; j=E[j].next) { // !!!!!
        int v = E[j].v, f = E[j].flow, c = E[j].cap;
        if(f < c && d[v] == d[u] + 1) {
            int aug = DFS(v, min(res, c-f));
            E[j].flow += aug; E[j^1].flow -= aug;
            res -= aug;
            if(res == 0) break; // !!!!!
        }
    }
    return flow - res;
}

int Dinic() {
    int MaxFlow = 0, CurFlow = 0;
```

```
    while(BFS()) {
        memcpy(cur, head, sizeof(head));
        while((CurFlow = DFS(s, INF)))
            MaxFlow += CurFlow;
    }
    return MaxFlow;
}

int main() {
    int u, v, c;
    scanf("%d%d%d%d", &n, &m, &s, &t);
    for(int i = 1; i <= m; i++) {
        scanf("%d%d%d", &u, &v, &c);
        AddEdge(u, v, c);
    }
    printf("%d", Dinic());
    return 0;
}
```

2.2 最小费用最大流

3 树

3.1 倍增 LCA

DoublingLCA.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Edge { int v, next; };

const int MAXN = 1e6, LOG = 20;
int n, q, s, e_ptr = 1, head[MAXN+10]; Edge E[(MAXN+10)<<1];
int dep[MAXN+10], anc[MAXN+10][LOG+1];

void add_edge(int u, int v) { E[++e_ptr] = (Edge) { v, head[u] }; head[u] = e_ptr; }
void add_pair(int u, int v) { add_edge(u, v); add_edge(v, u); }

void dfs(int u) {
    for(int i = 1; i <= LOG; i++)
        anc[u][i] = anc[anc[u][i-1]][i-1];
    for(int j=head[u]; j; j=E[j].next) {
        int v = E[j].v;
        if(v == anc[u][0]) continue;
        anc[v][0] = u; dep[v] = dep[u] + 1;
        dfs(v);
    }
}

int lca(int u, int v) {
    if(dep[u] < dep[v]) swap(u, v);
    for(int i = LOG; i >= 0; i--)
        if(dep[anc[u][i]] >= dep[v])
            u = anc[u][i];
    if(u == v) return u;
    for(int i = LOG; i >= 0; i--)
        if(anc[u][i] != anc[v][i])
            u = anc[u][i], v = anc[v][i];
    u = anc[u][0], v = anc[v][0];
    return u;
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

int main() {
    int u, v;
    n = readint(); q = readint(); s = readint();
    for(int i = 1; i <= n-1; i++) {
        u = readint(); v = readint();
        add_pair(u, v);
    }
    dep[s] = 1; dfs(s);
```

```
    while(q--) {
        u = readint(); v = readint();
        printf("%d\n", lca(u, v));
    }
    return 0;
}
```

3.2 欧拉序列求 LCA

EulerTourLCA.cpp

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e6;

struct Edge {
    int v, next;
};

int n, q, s, e_ptr = 1, dfs_clock, head[MAXN+10]; Edge E[(MAXN+10)<<1];
int dfn[MAXN+10], dfs_seq[MAXN+10], idx[MAXN+10], euler_seq[(MAXN+10)<<1], st[(MAXN+10)<<1][22];
/*
    dfn: dfs-clock of vertex u
    idx: the index of vertex u in euler-tour sequence
    dfs_seq: the dfs sequence
*/

void add_edge(int u, int v) {
    E[++e_ptr] = (Edge) { v, head[u] }; head[u] = e_ptr;
}

void add_pair(int u, int v) {
    add_edge(u, v); add_edge(v, u);
}

inline int readint() {
    int f=1, r=0; char c=getchar();
    while(!isdigit(c)) { if(c=='-')f=-1; c=getchar(); }
    while(isdigit(c)) { r=r*10+c-'0'; c=getchar(); }
    return f*r;
}

void dfs(int u, int fa) {
    euler_seq[++euler_seq[0]] = dfn[u] = ++dfs_clock;
    idx[u] = euler_seq[0]; dfs_seq[dfs_clock] = u;
    for(int j=head[u]; j; j=E[j].next) {
        int v = E[j].v;
        if(v == fa) continue;
        dfs(v, u);
        euler_seq[++euler_seq[0]] = dfn[u];
    }
}

void init_lca() {
    memset(st, 0x3f, sizeof(st));
```

```
    for(int i = 1; i <= euler_seq[0]; i++)
        st[i][0] = euler_seq[i];
    for(int j = 1; j <= 21; j++)
        for(int i = 1; i <= euler_seq[0] - (1<<j) + 1; i++) // bounds of sparse-table!
            st[i][j] = min(st[i][j-1], st[i + (1 << (j-1))][j-1]);
}

int query(int l, int r) {
    if(l > r) swap(l, r);
    int j;
    for(j = 0; (1 << (j+1)) <= (r-l+1); j++);
    return min(st[l][j], st[r - (1<<j) + 1][j]);
}

int lca(int u, int v) {
    return dfs_seq[query(idx[u], idx[v])];
}

int main() {
    int u, v;
    n = readint(); q = readint(); s = readint();
    for(int i = 1; i <= n-1; i++) {
        u = readint(); v = readint();
        add_pair(u, v);
    }
    dfs(s, -1); init_lca();
    while(q--) {
        u = readint(); v = readint();
        printf("%d\n", lca(u, v));
    }
    return 0;
}
```

3.3 树链剖分

3.4 点分治

4 单调数据结构

4.1 单调队列

4.2 单调栈

5 线段树

5.1 Lazy-Tag

5.2 动态开点线段树

5.3 可持久化线段树

5.4 二维线段树

6 平衡树

6.1 Treap

6.2 Splay

7 动态树

7.1 Link-cut Tree

8 字符串

8.1 KMP 字符串匹配

1-indexed

8.2 AC 自动机

0-indexed

ACAutomaton.cpp

```cpp
#include <bits/stdc++.h>
#define CLEAR(x) memset((x), 0, sizeof(x))
using namespace std;

const int SIGMA = 26, MAX_TEMP_LEN = 70, MAXN = 150,
MAX_LEN = 1e6, MAX_NODE = MAXN * MAX_TEMP_LEN;

int N, sz, ch[MAX_NODE + 10][SIGMA + 2], f[MAX_NODE + 10], last[MAX_NODE+10],
    val[MAX_NODE + 10], found_cnt[MAX_NODE+10];
char str[MAX_LEN+10], tpl[MAXN+10][MAX_TEMP_LEN+10];
unordered_map<string, int> ms;

inline int idx(char c) { return c - 'a' + 1; }

void insert(char *str) {
    int u = 0, len = strlen(str);
    for(int i = 0; i < len; i++) {
        int c = idx(str[i]);
        if(!ch[u][c]) ch[u][c] = ++sz;
        u = ch[u][c];
    }
    ms[string(str)] = u;
    ++val[u];
}

void get_fail() {
    queue<int> Q;
    f[0] = 0;
    for(int c = 1; c <= SIGMA; c++) if(ch[0][c]) {
        int v = ch[0][c];
        f[v] = last[v] = 0;
        Q.push(v);
    }
    while(!Q.empty()) {
        int u = Q.front(); Q.pop();
        for(int c = 1; c <= SIGMA; c++) {
            int v = ch[u][c];
            if(!v) {
                ch[u][c] = ch[f[u]][c];
                continue;
            }

            Q.push(v);

```

```
            int u2 = f[u];
            while(u2 && !ch[u2][c]) u2 = f[u2];
            f[v] = ch[u2][c];
            last[v] = val[f[v]] ? f[v] : last[f[v]];
        }
    }
}

void found(int u) {
    for(; u; u = last[u])
        found_cnt[u] += val[u];
}

void search(char *str) {
    int u = 0, len = strlen(str);
    for(int i = 0; i < len; i++) {
        int c = idx(str[i]);
        u = ch[u][c];
        if(val[u]) found(u);
        else if(last[u]) found(last[u]);
    }
}

inline void readstr(char *str) {
    char c=getchar(); int p=0;
    while(!isalnum(c) && !ispunct(c)) c = getchar();
    while(isalnum(c) || ispunct(c)) {
        str[p++] = c;
        c = getchar();
    }
    str[p++] = '\0';
}

int main() {
    while(true) {
        int ans = 0;
        sz = 0; CLEAR(ch); CLEAR(f); CLEAR(found_cnt);
        CLEAR(last); CLEAR(tpl); CLEAR(val); CLEAR(str);

        scanf("%d", &N); if(N == 0) break;
        for(int i = 1; i <= N; i++) {
            readstr(tpl[i]); insert(tpl[i]);
        }
        get_fail();

        readstr(str); search(str);

        for(int i = 0; i <= sz; i++)
            ans = max(ans, found_cnt[i]);
        printf("%d\n", ans);
        for(int i = 1; i <= N; i++)
            if(found_cnt[ms[string(tpl[i])]] == ans)
                printf("%s\n", tpl[i]);
    }
    return 0;
}
```

8.3 后缀数组

0-indexed

SuffixArray.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MAXLEN = 1e6, SIGMA = 100;

inline int idx(char c) {
    if(!c) return 0;
    else if(isdigit(c)) return c - '0' + 1;
    else if(isupper(c)) return c - 'A' + 1 + 10;
    else if(islower(c)) return c - 'a' + 1 + 10 + 26;
    else throw "Invalid Character";
}

struct SuffixArray {
    int sa[MAXLEN+10], rk[MAXLEN+10], buf[3][MAXLEN+10], height[MAXLEN+10], c[MAXLEN+10];
    void build_sa(char *s, int len) {
        int m = SIGMA + 10, n = len + 1, *x = buf[0], *y = buf[1];
        for(int i = 0; i < m; i++) c[i] = 0;
        for(int i = 0; i < n; i++) ++c[x[i] = idx(s[i])];
        for(int i = 1; i < m; i++) c[i] += c[i-1];
        for(int i = n-1; i >= 0; i--) sa[--c[x[i]]] = i;
        for(int k = 1; k <= n; k <<= 1) {
            int p = 0;
            for(int i = n-k; i < n; i++) y[p++] = i;
            for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
            for(int i = 0; i < m; i++) c[i] = 0;
            for(int i = 0; i < n; i++) ++c[x[y[i]]];
            for(int i = 1; i < m; i++) c[i] += c[i-1];
            for(int i = n-1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
            swap(x, y);
            p = 1, x[sa[0]] = 0;
            for(int i = 1; i < n; i++)
                x[sa[i]] = (y[sa[i]] == y[sa[i-1]] && y[sa[i] + k] == y[sa[i-1] + k] ? p-1 : p++);
            if(p >= n) break;
            m = p;
        }
        memcpy(rk, x, sizeof(rk));
        int k = 0;
        for(int i = 0; i < n; i++) {
            if(!rk[i]) continue;
            if(k) k--;
            int j = sa[rk[i]-1];
            while(s[i+k] == s[j+k]) k++;
            height[rk[i]] = k;
        }
    }
} SA;
inline void readstr(char* str) {
    char c=getchar(); int p=0;
    while(!isalnum(c) && !ispunct(c)) c=getchar();
    while(isalnum(c) || ispunct(c)) {
        str[p++] = c;
```

```
        c = getchar();
    }
    str[p++] = '\0';
}

int len;
char str[MAXLEN+10];

int main() {
    readstr(str); len = strlen(str);
    SA.build_sa(str, len);
    for(int i = 1; i <= len; i++)
        printf("%d ", SA.sa[i]+1);
    return 0;
}
```

9 Miscellaneous

9.1 ST 表

9.2 Fenwick Tree

9.3 左偏树

10 悬线法

10.1 Algorithm 1

10.2 Algorithm 2

11 莫队

11.1 普通莫队

11.2 带修改莫队

12 分块相关

12.1 分块

12.2 区间众数