线段树

P3373 【模板】线段树 2

如题，已知一个数列，你需要进行下面三种操作：

- 将某区间每一个数乘上 x
- 将某区间每一个数加上 x
- 求出某区间每一个数的和

输入格式

第一行包含三个整数 n,m,p。分别表示该数列数字的个数、操作的总个数和模数。

第二行包含n个用空格分隔的整数，其中第 i 个数字表示数列第 i项的初始值。

接下来m行每行包含若干个整数，表示一个操作，具体如下：

操作 1： 格式：`1 x y k` 含义：将区间 [x,y] 内每个数乘上 k

操作 2： 格式：`2 x y k` 含义：将区间 [x,y] 内每个数加上 k

操作 3： 格式：`3 x y` 含义：输出区间 [x,y] 内每个数的和对 p取模所得的结果。

输出格式

输出包含若干行整数，即为所有操作 3 的结果。

```
#include <iostream>
using namespace std;
#define int long long
const int N = 100010; int a[N], mod, n, m;
struct Node { int l, r, sum, add, mul; }tr[N * 4];
```

```
void pushup(int u) { tr[u].sum = (tr[u << 1].sum + tr[u << 1 | 1].sum) % mod; }
void eval(Node& node, int add, int mul) {
    node.sum = (node.sum * mul + (node.r - node.l + 1) * add) % mod;
    node.mul = (node.mul * mul) % mod;
    node.add = (node.add * mul + add) % mod;
}
void pushdown(int u) {
    eval(tr[u << 1], tr[u].add, tr[u].mul);
    eval(tr[u << 1 | 1], tr[u].add, tr[u].mul);
    tr[u].add = 0;
    tr[u].mul = 1;
}
void build(int l, int r, int u) {
    if (l == r) { tr[u] = { l, r, a[l], 0, 1 }; return; }
    tr[u] = { l, r, 0, 0, 1 };
    int mid = l + r >> 1;
    build(l, mid, u << 1); build(mid + 1, r, u << 1 | 1); pushup(u);
}
int query(int l, int r, int u) {
    if (tr[u].l >= l && tr[u].r <= r) { return tr[u].sum; }
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1, res = 0;
    if (l <= mid) { res += query(l, r, u << 1); }
    if (r > mid) { res += query(l, r, u << 1 | 1); }
    return res % mod;
}
void modify(int l, int r, int add, int mul, int u) {
    if (tr[u].l >= l && tr[u].r <= r) { eval(tr[u], add, mul); return; }
    pushdown(u);
    int mid = tr[u].l + tr[u].r >> 1;
    if (l <= mid) { modify(l, r, add, mul, u << 1); }
    if (r > mid) { modify(l, r, add, mul, u << 1 | 1); }
    pushup(u);
}
signed main() {
    cin >> n >> m >> mod;
    for (int i = 1; i <= n; i++) { cin >> a[i]; }
    build(1, n, 1);
    while (m--) {
        int op; cin >> op;
        if (op == 1) { int l, r, d; cin >> l >> r >> d; modify(l, r, 0, d, 1); }
        if (op == 2) { int l, r, d; cin >> l >> r >> d; modify(l, r, d, 1, 1); }
        if (op == 3) { int l, r; cin >> l >> r; cout << query(l, r, 1) << '\n';
}
    } return 0;
}
```

P1276 校门外的树（增强版）

我的理解：本题涉及到区间修改值（不是区间增加值）。

题目描述

校门外马路上本来从编号 00 到 L，每一编号的位置都有一棵树。有砍树者每次从编号 A到 B处连续砍掉每一棵树，就连树苗也不放过（记 `0 A B`，含 A和 B）。幸运的是还有植树者每次从编号 C到 D中凡是空穴（树被砍且还没种上树苗或树苗又被砍掉）的地方都补种上树苗（记 `1 C D`，含 C和 D）；问最终校门外留下的树苗多少棵？植树者种上又被砍掉的树苗有多少棵？

输入格式

第一行，两个正整数 L和 N，表示校园外原来有 L + 1棵树，并有 N次砍树或种树的操作。

以下 N行，每行三个整数，表示砍树或植树的标记和范围。

输出格式

共两行。第一行校门外留下的树苗数目，第二行种上又被拔掉的树苗数目。

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
//value表示：2为树苗 1为树木 0为没有树 -1为不能当做整体 可以省略一个标记
struct Node { int left, right, value; }tr[150000];
int l, n, ans1, ans2;  // ans1: 种上又被砍掉的树苗。 ans2: 最终留下的树苗。
void build(int left, int right, int u) {
    tr[u].left = left; tr[u].right = right; tr[u].value = 1;
    if (left == right) return;
    build(left, (left + right) >> 1, u << 1); build(((left + right) >> 1) + 1,
right, u << 1 | 1);
}
void updateCut(int left, int right, int u) {
    int mid = tr[u].left + tr[u].right >> 1; // 计算分段的位置
    if (tr[u].value != 1 && tr[u].value != -1) tr[u << 1].value = tr[u << 1 |
1].value = tr[u].value; //继承
    if (left <= tr[u].left && right >= tr[u].right) {
        if (tr[u].value == 0) return;  //没有树
        if (tr[u].value == 1) tr[u].value = 0; //有树木直接砍掉
        if (tr[u].value == 2) {
            tr[u].value = 0;  //树苗也要砍
            ans1 += (tr[u].right - tr[u].left) + 1;  //计算种上被砍掉的树苗
            ans2 = ans2 - (tr[u].right - tr[u].left + 1);  //计算留下的树苗
        }
        if (tr[u].value == -1) {//不能一起做就交给自己的左儿子和右儿子做
            tr[u].value = 0;
            updateCut(left, (left + right) >> 1, u << 1);
            updateCut(((left + right) >> 1) + 1, right, u << 1 | 1);
        }
    }
    else {
        if (right <= mid) updateCut(left, right, u << 1);
        else if (left > mid) updateCut(left, right, u << 1 | 1);
        else { updateCut(left, mid, u << 1); updateCut(mid + 1, right, u << 1 |
1); }
        if (tr[u << 1].value != tr[u << 1 | 1].value) tr[u].value = -1;
        else tr[u].value = tr[u << 1].value; //判断左右儿子是否相等，确定是否要整体一起
做
    }
}
void updatePlant(int left, int right, int u) {
    int mid = (tr[u].left + tr[u].right) >> 1;
```

```
    if (tr[u].value != 1 && tr[u].value != -1) tr[u << 1].value = tr[u << 1 |
1].value = tr[u].value; //同上
    if (left == tr[u].left && right == tr[u].right) {
        if (tr[u].value == 1 || tr[u].value == 2) return; //树木不要种!
        if (tr[u].value == 0) {
            tr[u].value = 2;
            ans2 += (tr[u].right - tr[u].left) + 1; //计算树苗的个数
        }
        if (tr[u].value == -1) { updatePlant(left, (left + right) >> 1, u << 1);
updatePlant(((left + right) >> 1) + 1, right, u << 1 | 1); }
    }
    else {
        if (right <= mid) updatePlant(left, right, u << 1);
        else if (left > mid) updatePlant(left, right, u << 1 | 1);
        else { updatePlant(left, mid, u << 1); updatePlant(mid + 1, right, u <<
1 | 1); }
        if (tr[u << 1].value != tr[u << 1 | 1].value) tr[u].value = -1;
        else tr[u].value = tr[u << 1].value; //同上
    }
}
signed main() {
    cin >> l >> n;
    build(0, l, 1);
    for (int i = 1; i <= n; i++) {
        int c, a, b; cin >> c >> a >> b;
        if (a > b) swap(a, b); if (c == 0) updateCut(a, b, 1); else
updatePlant(a, b, 1);
    }
    cout << ans2 << endl << ans1; return 0;
}
```

前缀树

我的理解：太简单，直接看代码

```
#include <bits/stdc++.h>
using namespace std;
struct Node { Node* nodes[26]; int cnt; };
Node* root = new Node();
void insert(string& s) {
    Node* p = root;
    for (int i = 0; i < s.size(); i++) {
        int idx = s[i] - 'a';
        if (p->nodes[idx] == nullptr) { p->nodes[idx] = new Node(); }
        p = p->nodes[idx];
        if (i == s.size() - 1) { p->cnt++; }
    }
}
int query(string& s) {
    Node* p = root; bool flag = true;
    for (int i = 0; i < s.size(); i++) {
        int idx = s[i] - 'a';
        if (p->nodes[idx] == nullptr) { flag = false; break; }
```

```
        p = p->nodes[idx];
    }
    if (flag) { return p->cnt; } else { return 0; }
}
int main() {
    int n; cin >> n;
    for (int i = 0; i < n; i++) {
        string op, s; cin >> op >> s;
        if (op == "I") { insert(s); }
        if (op == "Q") { cout << query(s) << endl; }
    }
    return 0;
}
```

DFS序

4310. 树的DFS

给定一棵 n 个节点的树。

节点的编号为 1~n，其中1号节点为根节点，每个节点的编号都大于其父节点的编号。

现在，你需要回答q个询问。

每个询问给定两个整数 ui, ki。

我们希望你用DFS算法来遍历根节点为 ui的子树。

我们规定，当遍历（或回溯）到某一节点时，下一个遍历的目标应该是它的未经遍历的子节点中编号最小的那一个子节点。

输入格式

第一行包含两个整数 n,q。

第二行包含 n−1n−1 个整数 p2,p3,...,pn，其中pi表示第i号节点的父节点的编号。

接下来q行，每行包含两个整数 ui,ki，表示一组询问。

输出格式

共q行，每组询问输出一行一个整数表示第ki个被遍历到的节点的编号。

如果第ki个被遍历到的节点不存在，则输出−1。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 200010;
vector<int> vers[N];
int n, q, sz[N], st[N], ntoi[N], iton[N], idx;
int dfs(int u) {
    iton[idx++] = u; sz[u] = 1; st[u] = true;
    for (int i = 0; i < vers[u].size(); i++) {
        int ne = vers[u][i];
        if (!st[ne]) { sz[u] += dfs(ne); }
    }
    return sz[u];
}
```

```
int main() {
    cin >> n >> q;
    for (int i = 2; i <= n; i++) {
        int p; cin >> p; vers[p].push_back(i); vers[i].push_back(p);
    }
    sz[1] = dfs(1);
    for (int i = 1; i <= idx; i++) { int num = iton[i]; ntoi[num] = i; }
    while (q--) {
        int u, k; cin >> u >> k;
        if (sz[u] < k) cout << -1 << '\n'; else cout << iton[ntoi[u] + k - 1] <<
'\n';
    } return 0;
}
```

平衡树

您需要写一种数据结构（可参考题目标题），来维护一些数，其中需要提供以下操作：

1. 插入数值 x。
2. 删除数值 x(若有多个相同的数，应只删除一个)。
3. 查询数值 x 的排名(若有多个相同的数，应输出最小的排名)。
4. 查询排名为 x 的数值。
5. 求数值 x 的前驱(前驱定义为小于 x 的最大的数)。
6. 求数值 x 的后继(后继定义为大于 x 的最小的数)。

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
#include <bits/stdc++.h>
using namespace std;
template <typename T>
using ordered_multiset = tree<T, null_type, less_equal<T>, rb_tree_tag,
tree_order_statistics_node_update>;
int main() {
    ordered_multiset<int> st; int n; cin >> n;
    while (n--) {
        int op, x;
        cin >> op >> x;
        if (op == 1) st.insert(x);
        if (op == 2) st.erase(st.upper_bound(x));
        if (op == 3) cout << st.order_of_key(x) + 1 << endl;
        if (op == 4) cout << *st.find_by_order(x - 1) << endl;
        if (op == 5) cout << *--st.upper_bound(x) << endl;
        if (op == 6) cout << *st.lower_bound(x) << endl;
    } return 0;
}
```

最短路

dijkstra堆优化版

图中，dijkstra算法可以快速求出一个点到其余点的最短路径。

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> PII;
const int N = 150010;
vector<int> vers[N], edges[N]; int n, m, d[N]; bool st[N];
void add(int a, int b, int c) { vers[a].push_back(b); edges[a].push_back(c); }
void dijkstra() {
    memset(d, 0x3f, sizeof d); priority_queue<PII, vector<PII>, greater<PII> >
heap;
    d[1] = 0; heap.push({ 0, 1 });
    while (heap.size()) {
        PII t = heap.top(); heap.pop();
        int x = t.second; if (st[x]) continue; st[x] = true;
        for (int i = 0; i < vers[x].size(); i++) {
            int y = vers[x][i], w = edges[x][i];
            if (d[y] > d[x] + w) { d[y] = d[x] + w; heap.push({ d[y], y }); }
        }
    }
}
int main() {
    cin >> n >> m;
    while (m--) { int a, b, c; cin >> a >> b >> c; add(a, b, c); }
    dijkstra();
    if (d[n] == 0x3f3f3f3f) cout << -1 << '\n';
    else cout << d[n] << '\n'; return 0;
}
```

floyd算法

O(N^3)的时间复杂度求出图中任意两点的最短路径。

```
#include<iostream>
using namespace std;
const int N = 210, INF = 0x3f3f3f3f; int dist[N][N], n, m, k;
void floyd() {
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}
int main() {
    cin >> n >> m >> k;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j) { dist[i][j] = 0; }
            else { dist[i][j] = INF; }
        }
    }
    for (int i = 0; i < m; i++) {
        int x, y, w; cin >> x >> y >> w;
```

```
        dist[x][y] = min(dist[x][y], w);
    }
    floyd();
    for (int i = 0; i < k; i++) {
        int x, y; cin >> x >> y;
        if (dist[x][y] > INF / 2) { cout << "impossible" << endl; }
        else { cout << dist[x][y] << endl; }
    }
    return 0;
}
```

SPFA

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100010, INF = 0x3f3f3f3f; int e[N], ne[N], w[N], h[N], idx,
dist[N]; bool st[N];
void add(int a, int b, int c) { e[idx] = b; ne[idx] = h[a]; w[idx] = c; h[a] =
idx++; }
void spfa() {
    dist[1] = 0; st[1] = true; queue<int> q; q.push(1);
    while (q.size()) {
        int t = q.front(); q.pop(); st[t] = false;
        for (int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];
                if (!st[j]) { st[j] = true; q.push(j); }
            }
        }
    }
}
int main() {
    memset(h, -1, sizeof h); memset(dist, 0x3f, sizeof dist);
    int n, m; cin >> n >> m;
    while (m--) { int a, b, c; cin >> a >> b >> c; add(a, b, c); }
    spfa();
    if (dist[n] == INF) cout << "impossible"; else cout << dist[n]; return 0;
}
```

并查集

837. 连通块中点的数量

给定一个包含n个点（编号为1~n）的无向图，初始时图中没有边。

现在要进行m个操作，操作共有三种：

1. `C a b`，在点a和点b之间连一条边，a和b可能相等；
2. `Q1 a b`，询问点a和点b是否在同一个连通块中，a和b可能相等；
3. `Q2 a`，询问点a所在连通块中点的数量；

输入格式

第一行输入整数n和m。接下来m行，每行包含一个操作指令，指令为 `C a b`，`Q1 a b` 或 `Q2 a` 中的一种。

输出格式

对于每个询问指令 `Q1 a b`，如果 aa 和 bb 在同一个连通块中，则输出 `Yes`，否则输出 `No`。

对于每个询问指令 `Q2 a`，输出一个整数表示点 aa 所在连通块中点的数量。每个结果占一行。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100010; int p[N], g[N];
int find(int x) { if (x != p[x]) p[x] = find(p[x]); return p[x]; }
int main() {
    int n, m; cin >> n >> m;
    for (int i = 1; i <= n; i++) { p[i] = i; g[i] = 1; }
    while (m--) {
        string s; cin >> s;
        if (s == "C") {
            int a, b; cin >> a >> b;
            if (find(a) != find(b)) { g[find(b)] += g[find(a)]; p[find(a)] =
find(b); }
        }
        else if (s == "Q1") {
            int a, b; cin >> a >> b;
            if (find(a) != find(b)) { cout << "No" << endl; }
            else { cout << "Yes" << endl; }
        }
        else { int x; cin >> x; cout << g[find(x)] << endl; }
    } return 0;
}
```

拓扑排序

可达性统计

给定一张N个点M条边的有向无环图，分别统计从每个点出发能够到达的点的数量。

输入格式

第一行两个整数N,M接下来M行每行两个整数 x,y表示从x到y的一条有向边。

输出格式

输出共N行，表示每个点能够到达的点的数量。

数据范围

1≤N,M≤30000

```
#include <bits/stdc++.h>
using namespace std;
const int N = 30010; int d[N], q[N], hh, tt = -1, sorted_idx[N], n, m, idx;
vector<int> vers[N], res;
bitset<N> f[N];  // 前一个<N>是第一维数组，后一个[N]是第二维数组。
```

```
void topsort() {  // 这个函数是拓扑排序的所有实现，包含变量d,q,hh,tt,sorted_idx,idx。
    for (int i = 1; i <= n; i++) { if (!d[i]) { q[++tt] = i; sorted_idx[idx++] =
i; } }
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = 0; i < vers[t].size(); i++) {
            int ne = vers[t][i];
            if (--d[ne] == 0) { q[++tt] = ne; sorted_idx[idx++] = ne; }
        }
    }
}
int main() {
    cin >> n >> m;
    while (m--) { int a, b; cin >> a >> b; vers[a].push_back(b); d[b]++; }
    topsort();
    for (int i = idx - 1; i >= 0; i--) {
        int id = sorted_idx[i]; f[id][id] = 1;
        for (int j = 0; j < vers[id].size(); j++) {
            int ne_idx = vers[id][j];
            f[id] |= f[ne_idx];
        }
    }
    for (int i = 1; i <= n; i++) cout << f[i].count() << endl; return 0;
}
```

二分图

染色法判定二分图

给定一个n个点m条边的无向图，图中可能存在重边和自环。

请你判断这个图是否是二分图。

输入格式

第一行包含两个整数n和m。

接下来m行，每行包含两个整数u和 v，表示点u和点v之间存在一条边。

输出格式

如果给定图是二分图，则输出 `Yes`，否则输出 `No`。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 200010; int e[N], ne[N], h[N], idx, color[N]; bool res = true;
void add(int a, int b) { e[idx] = b; ne[idx] = h[a]; h[a] = idx++; }
void dfs(int u, int col) {
    color[u] = col;
    for (int i = h[u]; i != -1; i = ne[i]) {
        if (!color[e[i]]) dfs(e[i], 3 - col);
        else if (color[e[i]] == col) { res = false; return; }
    }
}
int main() {
    memset(h, -1, sizeof h); int n, m; cin >> n >> m;
```

```
    while (m--) { int a, b; cin >> a >> b; add(a, b); add(b, a); }
    for (int i = 1; i <= n; i++) {
        if (!color[i]) { dfs(i, 1); }
        if (!res) break;
    }
    if (res) cout << "Yes";
    else cout << "No"; return 0;
}
```

二分图的最大匹配

给定一个二分图，其中左半部包含n1个点（编号1~n1），右半部包含n2个点（编号1~n），二分图共包含m条边。

数据保证任意一条边的两个端点都不可能在同一部分中。

请你求出二分图的最大匹配数。

输入格式

第一行包含三个整数n1、n2和m。

接下来m行，每行包含两个整数u和v，表示左半部点集中的点u和右半部点集中的点v之间存在一条边。

输出格式

输出一个整数，表示二分图的最大匹配数。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 510, M = 100010; int e[M], ne[M], h[N], idx, match[N]; bool st[N];
void add(int a, int b) { e[idx] = b; ne[idx]= h[a]; h[a] = idx++; }
bool find(int x) {
    for (int i = h[x]; i != -1; i = ne[i]) {
        if (!st[e[i]]) {
            st[e[i]] = true;
            if (!match[e[i]]) { match[e[i]] = x; return true; }
            else if (match[e[i]] && find(match[e[i]])) { match[e[i]] = x; return
true; }
        }
    }
    return false;
}
int main() {
    memset(h, -1, sizeof h);
    int n1, n2, m, res = 0; cin >> n1 >> n2 >> m;
    while (m--) { int u, v; cin >> u >> v; add(u, v); }
    for (int i = 1; i <= n1; i++) { memset(st, false, sizeof st); if (find(i))
res++; }
    cout << res; return 0;
}
```

最近公共祖先

给定一棵包含n个节点的有根无向树，节点编号互不相同，但不一定是1~n。

有m个询问，每个询问给出了一对节点的编号x和y，询问x与y的祖孙关系。

输入格式

输入第一行包括一个整数 表示节点个数；

接下来n行每行一对整数a和b，表示a和b之间有一条无向边。如果b是-1，那么a就是树的根；

第n+2行是一个整数m表示询问个数；

接下来m行，每行两个不同的正整数x和y，表示一个询问。

输出格式

对于每一个询问，若x是y的祖先则输出1，若y是x的祖先则输出2，否则输出0。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 4e4 + 10; int n, s, dep[N], fa[N][20]; vector<int> vers[N];
void dfs(int u, int father) {
    dep[u] = dep[father] + 1;
    fa[u][0] = father;  // u结点往上走2^0步的祖先是father
    for (int i = 1; i <= 19; i++) { fa[u][i] = fa[fa[u][i - 1]][i - 1]; }
    for (int& v : vers[u]) { if (v != father) { dfs(v, u); } }
}
int lca(int a, int b) {
    bool hasSwap = false;
    if (dep[a] < dep[b]) swap(a, b), hasSwap = true;
    for (int i = 19; i >= 0; i--) {  //必须倒着遍历
        if (dep[fa[a][i]] >= dep[b]) { a = fa[a][i]; }
    }
    if (a == b) { if (hasSwap) return 1; else return 2; }
    return 0;
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        int a, b; cin >> a >> b;
        if (b == -1) s = a; else vers[a].push_back(b), vers[b].push_back(a);
    }
    dfs(s, 0);
    int m; cin >> m;
    while (m--) { int a, b; cin >> a >> b; cout << lca(a, b) << endl; } return
0;
}
```

KMP

假定有字符串A和B。KMP能在O(N)的时间内求得字符串A中所有以第1个字符开始的子串在字符串B中的最大匹配长度。

KMP字符串

给定一个字符串 S，以及一个模式串 P，所有字符串中只包含大小写英文字母以及阿拉伯数字。

模式串 P 在字符串 S 中多次作为子串出现。

求出模式串 P 在字符串 S 中所有出现的位置的起始下标。

```
#include <iostream>
#include <string>
using namespace std;

const int N = 1000010;
int ne[N];
int main() {
    int n, m;
    string small, big;
    cin >> m >> small >> n >> big;
    small = ' ' + small, big = ' ' + big;

    for (int i = 2, j = 0; i <= m; i++) {
        while (j && small[i] != small[j + 1]) j = ne[j];
        if (small[i] == small[j + 1]) j++;
        ne[i] = j;
    }

    for (int i = 1, j = 0; i <= n; i++) {
        while (j && big[i] != small[j + 1]) j = ne[j];
        if (big[i] == small[j + 1]) j++;
        if (j == m) {
            cout << i - j << ' ';
            j = ne[j];
        }
    }
    return 0;
}
```