МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
“Национальный исследовательский университет ИТМО”

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине
“АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ”
Базовые задачи.

выполнил:
Студент группы P32311
Птицын Максим Евгеньевич
Преподаватели:
Косяков М.С.
Тараканов Д.С.

г. Санкт-Петербург
2023 г.

Содержание

1 329. Галактическая история

Условие задачи:
там чет лютый лонгрид

Пояснение к примененному алгоритму:
ну типа на каждом запросе подниматься вверх по родителям - TL, хранить у каждой вершины список её родителей - ML, поэтому LCA./
препроцессинг: обход в глубину, заходя в каждую вершину записываю её глубину, заношу в лист с порядком обхода, выходя из её ребёнка так же заношу в лист./
так же записываю позицию каждой вершины в массив с позициями, так что для каждой i вершины list[pos[i]] = i, причём гарантировано, что pos[i] указывает на первое включение вершины на пути обхода, и строю дерево отрезков из листа с порядком обхода./
далее на каждый запрос прохожу рекурсивно по дереву отрезков, пока не найду общего предка./ Сложность алгоритма: Препроцессинг O(N) + Запрос O($log(N)$) * L количество запросов/

Код:

```
typedef vector<vector<int> > graph;
typedef vector<int>::const_iterator const_graph_iter;


vector<int> lca_h, lca_dfs_list, lca_pos, lca_tree;
vector<char> lca_dfs_used;
unordered_map<int, int> indexes;


void lca_dfs(const graph &g, int v, int h = 1) {
    lca_dfs_used[v] = true;
    lca_h[v] = h;
    lca_dfs_list.push_back(v);
    for (const_graph_iter i = g[v].begin(); i != g[v].end(); ++i)
        if (!lca_dfs_used[*i]) {
            lca_dfs(g, *i, h + 1);
            lca_dfs_list.push_back(v);
        }
}

void lca_build_tree(int i, int l, int r) {
    if (l == r)
        lca_tree[i] = lca_dfs_list[l];
    else {
        int m = (l + r) >> 1;
        lca_build_tree(i + i, l, m);
        lca_build_tree(i + i + 1, m + 1, r);
        if (lca_h[lca_tree[i + i]] < lca_h[lca_tree[i + i + 1]])
            lca_tree[i] = lca_tree[i + i];
        else
            lca_tree[i] = lca_tree[i + i + 1];
    }
}

void lca_prepare(const graph &g, int root) {
    int n = (int) g.size();
    lca_h.resize(n);
    lca_dfs_list.reserve(n * 2);
    lca_dfs_used.assign(n, 0);

    lca_dfs(g, root);

    int m = (int) lca_dfs_list.size();
    lca_tree.assign(lca_dfs_list.size() * 4 + 1, -1);
    lca_build_tree(1, 0, m - 1);

    lca_pos.assign(n, -1);
    for (int i = 0; i < m; ++i) {
        int v = lca_dfs_list[i];
        if (lca_pos[v] == -1)
            lca_pos[v] = i;
    }
}

int lca_tree_min(int i, int sl, int sr, int l, int r) {
    if (sl == l && sr == r)
        return lca_tree[i];
    int sm = (sl + sr) >> 1;
    if (r <= sm)
        return lca_tree_min(i + i, sl, sm, l, r);
    if (l > sm)
        return lca_tree_min(i + i + 1, sm + 1, sr, l, r);
    int ans1 = lca_tree_min(i + i, sl, sm, l, sm);
    int ans2 = lca_tree_min(i + i + 1, sm + 1, sr, sm + 1, r);
    return lca_h[ans1] < lca_h[ans2] ? ans1 : ans2;
}

int lca(int a, int b) {
    int left = lca_pos[a],
```

```
            right = lca_pos[b];
    if (left > right) swap(left, right);
    return lca_tree_min(1, 0, (int) lca_dfs_list.size() - 1, left, right);
}

int main() {
    size_t N;
    cin >> N;
    int max_v = 0;
    unordered_multimap<int, int> vertexes;
    for (size_t i = 0; i < N; i++) {
        int a, b;
        cin >> a >> b;
        max_v = max(max_v, a);
        max_v = max(max_v, b);
        vertexes.emplace(b, a);
    }
    graph g(max_v + 2);
    int root = max_v + 1;
    auto iter = vertexes.begin();
    while (!vertexes.empty()) {
        vector<int> childs;
        int parent = (*iter).first;
        if (parent == -1) parent = max_v + 1;
        while (iter != vertexes.end()) {
            int child = (*iter).second;
            childs.push_back(child);
            vertexes.erase(iter);
            if (parent == -1) parent = max_v + 1;
            iter = vertexes.find(parent);
            if (parent == max_v + 1) parent = -1;
        }
        if (parent == -1) parent = max_v + 1;
        g[parent] = childs;
        indexes.emplace(parent, g.size() - 1);
        iter = vertexes.begin();
    }
    lca_prepare(g, root);

    size_t L;
    cin >> L;
    for (size_t i=0; i< L; i++) {
        int a,b;
        cin >> a >> b;
        int v = lca(a, b); // ответ на запрос
        if (v == a) cout << 1 << endl;
        else if (v==b) cout << 2 << endl;
        else cout << 0 << endl;
    }
}
```

2 1450. Российские газопроводы

Условие задачи:
nen/

Пояснение к примененному алгоритму:
задача:найти путь наибольшей стоимости из одной вершины в другую./ алгоритм: модифицированный беллман-форд с выходом из обхода, если на каком-то шаге не было увеличено стоимости пути/

Сложность алгоритма: O(h*M), где h - глубина ориентированного дерева, подвешенного за стартовую вершину./

Код:

```
int n, m;
std::cin >> n >> m;
std::vector<std::vector<int> > edges(m, std::vector<int>(3));
for (int i = 0; i < m; ++i) {
    std::cin >> edges[i][0] >> edges[i][1] >> edges[i][2];
}
std::vector<int> F(n + 1, -1);
int s, f;
std::cin >> s >> f;
F[s] = 0;
bool stop = false;
for (int k = 1; k < n && !stop; ++k) {
    stop = true;
    for (auto &edge: edges) {
        int start = edge[0];
        int finish = edge[1];
        int weight = edge[2];
        if (F[start] != -1 && F[start] + weight > F[finish]) {
            F[finish] = F[start] + weight;
            stop = false;
        }
    }
}
if (F[f] == -1) std::cout << "No solution";
else std::cout << F[f];
return 0;
```