Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №4**

по «Алгоритмам и структурам данных»

Timus

Выполнил:

Студент группы P32092

Голиков А.С.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача №1329 «Галактическая история»

Пояснение к примененному алгоритму:

Задача - определить, является ли одна из пары вершин предком другой. Давайте решим более общую задачу - найдем ближайшего общего предка этих двух вершин, и очевидно, он совпадет с одной из наших вершин в том и только том случае, если она является предком второй.

Для этого давайте обойдем дерево в глубину и запишем порядок, в котором посещаем вершины, причем, когда идем вверх, тоже будем их записывать.

Так, на любом отрезке массива, содержащем обе эти вершины, будет и их общий предок, так как отрезок это буквально путь по графу между концами этого отрезка.

Посмотрим на отрезок между первыми вхождениями нужных нам вершин. Очевидно, общим предком будет та вершина из этого отрезка, которая находится ближе всех к корню.

Это задача поиска минимума на отрезке, а ее очень хорошо решает дерево отрезков.

Асимптотика - O(L(N + logN))

Код решения:

#include <iostream>

#include <unordered\_map>

#include <vector>

#include <unordered\_set>

#include <cmath>

using namespace std;

vector<int> order;

unordered\_map<int, int> first;

unordered\_map<int, int> height;

unordered\_map<int, vector<int>> graph;

int n;

int index = 0;

void dfs(unordered\_set<int> \*visited, int id, int h = 0) {

order.push\_back(id);

(\*visited).insert(id);

height[id] = h;

first[id] = index++;

for (auto c: graph[id]) {

if ((\*visited).find(c) == (\*visited).end()) {

dfs(visited, c, h + 1);

order.push\_back(id);

index++;

}

}

}

int minimal(const vector<int> &a, int v, int l, int r, int lv, int rv) {

if (l >= rv || lv >= r)

return n;

if (lv >= l && rv <= r)

return a[v];

int m = (lv + rv) / 2;

int sl = minimal(a, 2 \* v, l, r, lv, m);

int sr = minimal(a, 2 \* v + 1, l, r, m, rv);

return min(sl, sr);

}

int main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

cout.tie(nullptr);

cin >> n;

unordered\_set<int> visited;

int root;

for (int i = 0; i < n; ++i) {

int child, parent;

cin >> child >> parent;

if (parent == -1) root = child;

if (graph.find(parent) == graph.end()) {

graph[parent] = **{**child**}**;

} else {

graph[parent].push\_back(child);

}

}

dfs(&visited, root);

int size = (int) order.size();

int p = (int) pow(2, (int) ceil(log2(size)));

vector<int> tree(2 \* p);

for (int i = 0; i < size; ++i) {

tree[p + i] = height[order[i]];

}

for (int i = p - 1; i > 0; --i) {

tree[i] = min(tree[2 \* i], tree[2 \* i + 1]);

}

int l;

cin >> l;

for (int i = 0; i < l; ++i) {

int a, b;

cin >> a >> b;

int left = p + min(first[a], first[b]);

int right = p + max(first[a], first[b]) + 1;

int ans = minimal(tree, 1, left, right, p, 2 \* p);

if (ans == height[a]) {

cout << 1 << "\n";

} else if (ans == height[b]) {

cout << 2 << "\n";

} else {

cout << 0 << "\n";

}

}

return 0;

}

Задача №1806 «Мобильные телеграфы»

Пояснение к примененному алгоритму:

Задача - определить кратчайший путь в графе с нетривиальными ребрами)))

Эту задачу решает алгоритм Дейкстры, но возникает проблема - слишком большое N. Для обычного построения графа нужно O(N^2) операций, чтобы проверить все ребра. Тут так нельзя, да к тому же хранение всего графа займет слишком много памяти

Ок - не будем строить. Сразу заметим, что так как длина каждой id вершины 10, у нее может быть не более, чем 9\*10 + С\_10:^2 = 135 соседей, так на шаге Дейкстры их и проверять.

Асимптотика - O(mlogn)

Код решения:

#include <iostream>

#include <vector>

#include <queue>

#include <unordered\_map>

#include <unordered\_set>

#define inf 32767

using namespace std;

vector<int> weight(10);

int n;

unordered\_map<string, int> vertex;

int dist(const string &s1, const string &s2) {

int pref = 0;

for (; s1[pref] == s2[pref];) {

++pref;

}

return weight[pref];

}

pair<int, vector<int>> dijkstra(const string& start, const string& finish) {

unordered\_set<string> visited;

priority\_queue<pair<int, string>, vector<pair<int, string>>, greater<>> q;

unordered\_map<string, pair<int, string>> way;

way[start] = {0, ""};

q.emplace(0, start);

while (!q.empty()) {

string v = q.top().second;

if (v == finish) {

break;

}

q.pop();

for (int i = 0; i < 10; ++i) {

for (int j = i + 1; j < 10; ++j) {

string curr = v;

if (v[i] != v[j]) {

curr[i] = v[j];

curr[j] = v[i];

if (vertex.find(curr) != vertex.end()) {

int len = dist(v, curr);

if (visited.find(curr) == visited.end()) {

if ((way.find(curr) == way.end()) || (way[v].first + len < way[curr].first)) {

way[curr] = make\_pair(way[v].first + len, v);

q.emplace(way[curr].first, curr);

}

}

}

}

}

}

for (int i = 0; i < 10; ++i) {

for (char j = '0'; j <= '9'; ++j) {

if (v[i] != j) {

string curr = v;

curr[i] = j;

if (vertex.find(curr) != vertex.end()) {

int len = dist(v, curr);

if (visited.find(curr) == visited.end()) {

if ((way.find(curr) == way.end()) || (way[v].first + len < way[curr].first)) {

way[curr] = make\_pair(way[v].first + len, v);

q.emplace(way[curr].first, curr);

}

}

}

}

}

}

visited.insert(v);

}

if (way.find(finish)==way.end()) {

return {inf, {}};

}

vector<int> ans = **{**vertex[finish]**}**;

string pred = finish;

while (pred != start) {

pred = way[pred].second;

ans.push\_back(vertex[pred]);

}

return {way[finish].first, ans};

}

int main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

cout.tie(nullptr);

cin >> n;

for (int i = 0; i < 10; ++i) {

cin >> weight[i];

}

string start, finish;

for (int i = 0; i < n; ++i) {

string s;

cin >> s;

if (i == 0){

start = s;

}

if (i == n - 1){

finish = s;

}

vertex[s] = i + 1;

}

pair<int, vector<int>> ret = dijkstra(start, finish);

if (ret.first == inf) {

cout << -1;

return 0;

}

int size = ret.second.size();

cout << ret.first << "\n";

cout << size << "\n";

for (int i = 0; i < size; ++i) {

cout << ret.second[size - 1 - i] << " ";

}

return 0;

}