Университет ИТМО
Кафедра ВТ

Задачи 4 (1 - 5)
Алгоритмы и Структуры Данных

Выполнил: Федоров Сергей
Группа: P3212

Санкт-Петербург
2020 г.

- # Задача 1 - Раскраска Карты - 1080

Представим страны как граф. Вершины - страны, ребра - границы. Наша задача покрасить вершины граф. Так как у нас два цвета, то граф должен будет получится двудольным (или нет). Поэтому красим граф обход в ширину.

O(|V| + |E|)

Код:

```
//
// Created by Sergey Fedorov on 28/05/2020.
//

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

const int START_COLOR = 0;

bool btf(int root, vector<vector<int>> &edge, vector<int> &color) {
    queue<int> to_visit;

    to_visit.push(root);
    color[root] = START_COLOR;

    bool finished = false;
    bool success = true;

    while (!to_visit.empty() && !finished) {
        int from = to_visit.front(); to_visit.pop();

        for (int i = 0; i < edge[from].size(); i++) {
            int to = edge[from][i];

            if (color[from] == color[to]) {
                success = false;
                finished = true;
            } else if (color[to] == -1) {
                color[to] = !color[from];
                to_visit.push(to);
            }
        }
    }

    return success;
}

int main(){
    int n; cin >> n;
    vector<int> colors(n, -1);
    vector<vector<int>> borders(n);

    for (int a = 0; a < n; a++) {
        int b;
        cin >> b;
        while (b != 0) {
                b--;
                borders[a].push_back(b);
```

```
                borders[b].push_back(a);
                cin >> b;
            }
        }

        bool success = true;
        for (int i = 0; i < n; i++) if (colors[i] == -1) {
            bool result = btf(i, borders, colors);
            success &= result;
            if (!result) break;
        }

        if (success) {
            for (int i = 0; i < n; i++) cout << colors[i];
        } else {
            cout << "-1";
        }
}
```

- **Задача 2 - Российские Газопроводы - 1450**

Здесь нам надо найти наибольший путь. Это обратно поиску наименьшего пути. У нас есть пару алгоритмов на выбор, но алгоритм Форда-Беллмана показался самым легким, поэтому возьмем его.

Есть пару способов находить наибольший путь вместо наименьшего. В данном случае делаем веса отрицательными (благо Алг Ф-Б умеет с ними работать).

(плюс есть вот такой вот пост, весьма оправдывающий мой выбор https://stackoverflow.com/questions/3447566/dijkstras-algorithm-in-c)

`O(|V| * |E|)`

Код:

```
//
// Created by Sergey Fedorov on 28/05/2020.
//

#include <iostream>
#include <vector>

using namespace std;

#define INF (INT_MAX/2)

struct pipe {
    int a, b, meta_dist;
};

int main() {
    int n, m;
    cin >> n >> m;

    vector<pipe> pipes(m);

    for (int i = 0; i < m; ++i) {
        int a, b, p;
        cin >> a >> b >> p;
        pipes[i] = {a - 1, b - 1,  -p};
    }
```

```
    int start, finish;
    cin >> start >> finish;
    start--;
    finish--;

    vector<int> sum_dist(n, INF);
    sum_dist[start] = 0;

    // Bellman-Ford algorithm (without check for cycles)
    for (int i = 0; i < n - 1; ++i) for (int j = 0; j < m; ++j) {
        if (sum_dist[pipes[j].a] != INF &&
            sum_dist[pipes[j].b] > sum_dist[pipes[j].a] + pipes[j].meta_dist) {
            sum_dist[pipes[j].b] = sum_dist[pipes[j].a] + pipes[j].meta_dist;
        }
    }

    if (sum_dist[finish] != INF) cout << -sum_dist[finish]; else cout << "No solution";
}
```

- **Задача 3 - Network - 1160**

Строим граф соединения хабов. Потом, понимаем что условие задачи - перефразированное определение MST. Выбираем алгоритм построения MST на свой вкус, строим.

P.S. Долго не мог понять почему не выводится правильный ответ на открытый тест…. Оказалось открытый тест - неправильный.

O(|E| * log(|E|))

Код:

```
//
// Created by Sergey Fedorov on 28/05/2020.
//

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct connection {
    int a, b, length;
    bool used = false;

    connection()= default;

    connection(int a, int b, int length){
        this->a = a;
        this->b = b;
        this->length = length;
    }

};

bool connection_comprar(connection fst, connection snd){
    return fst.length < snd.length;
```

```
}

int find_parent(vector<int> &parents, int x){
    if (x ≠ parents[x]) parents[x] = find_parent(parents, parents[x]);
    return parents[x];
}

int main(){
    int n, m;
    cin >> n >> m;

    vector<connection> cs(m);
    vector<int> ranks(n + 1, 0);
    vector<int> parents(n + 1);
    for (int i = 0; i ⩽ n; ++i) parents[i] = i;

    int longest_conn = 0;
    int out = 0;

    for (int i = 0; i < m; ++i) {
        int a, b, l;
        cin >> a >> b >> l;
        a--; b--;
        cs[i] = connection(a, b, l);
    }

    sort(cs.begin(), cs.end(), connection_comprar);

    for (int i = 0; i < m; ++i) {
        int point_a = cs[i].a;
        int point_b = cs[i].b;

        int parent_a = find_parent(parents, point_a);
        int parent_b = find_parent(parents, point_b);

        if (parent_a ≠ parent_b) {
            longest_conn = max(longest_conn, cs[i].length);

            out++;
            cs[i].used = true;

            if      (ranks[parent_a] > ranks[parent_b]) parents[parent_b] = parent_a;
            else if (ranks[parent_b] > ranks[parent_a]) parents[parent_a] = parent_b;
            else if (ranks[parent_a] == ranks[parent_b]) {
                parents[parent_b] = parent_a;
                ranks[parent_a]++;
            }

        }
    }

    cout << longest_conn << endl;
    cout << out << endl;

    for (connection conn : cs) if (conn.used) cout << conn.a + 1 << " " << conn.b + 1 <<
endl;
}
```

- **Задача 4 - Currency Exchange - 1162**

Задача практически аналогична задаче 1160, однако тут по другому считаем расстояние, а так же в конце выбираем не какую-то конкретную вершину, а ищем хотя бы одну в которой получается прирост по финансам.

(
For example, if you want to exchange 100 US Dollars into Russian Rubles at the exchange point, where the exchange rate is 29.75, and the commission is 0.39 you will get (100 - 0.39) * 29.75 = 2963.3975RUR.
) ☹️

`O(|V| * |E|)`

Код:

```
//
// Created by Sergey Fedorov on 30/05/2020.
//

#include <iostream>
#include <vector>

#define EPSILON 1e-5

using namespace std;

struct exchange_point {
    int cur_a, cur_b;
    double rab, cab;
};

int main(){
    int n, m, s;
    double v;

    cin >> n >> m >> s >> v;

    vector<exchange_point> ex_ps(2 * m);
    vector<double> currencies(n + 1, 0);

    for (int i = 0; i < 2 * m; i += 2) {
        int a, b;
        double rab, cab, rba, cba;
        cin >> a >> b >> rab >> cab >> rba >> cba;

        ex_ps[i    ] = {a, b, rab, cab};
        ex_ps[i + 1] = {b, a, rba, cba};
    }

    currencies[s] = v;

    // Need to twist the algo, to find the biggest path
    // This time just negation didn't work :(
    for (int i = 0; i < n - 1; i++) for (auto &ex_p : ex_ps)
        currencies[ex_p.cur_b] = max(currencies[ex_p.cur_b], (currencies[ex_p.cur_a] -
ex_p.cab) * ex_p.rab);
```

```
    bool wealth_growth = false;
    for (auto &ex_p : ex_ps)
        wealth_growth |= (currencies[ex_p.cur_a] - ex_p.cab) * ex_p.rab >
currencies[ex_p.cur_b] + EPSILON;

    if (wealth_growth) cout << "YES"; else cout << "NO";
}
```

• Задача 5 - Мобильные телеграфы - 1806

Раздел - “Графы”. Значит строим граф))

Построение графа: перебираем возможные комбинации по значениям и по перестановкам. Если вдруг получаем совпадение с уже известными телеграфами, добавляем связь, одновременно проверяя задержку по времени, сравнивая префиксы.

O(|V|)

После того как построили граф, находим кратчайший путь до Чапаева. Пытался решить алгоритмом Беллмана-Форда, но так как граф - не плотный (или я что-то не так сделал :)), то алгоритм - неэффективный. Взял алгоритм Дейкстры, зашло.

O(|V| * log(|V|) + |E| * log(|V|))

Код:

```
//
// Created by Sergey Fedorov on 05/06/2020.
//

#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>

using namespace::std;


#define INF (LONG_MAX/2)
#define WIDTH 10

int get_digit(long long num, int pos){
    long long denominator = 1;
    for (int i = 0; i < pos; ++i) denominator *= 10;
    return (int) (num / denominator % 10);
}

long long set_digit(long long num, int pos, int value){
    long long power = 1;
    for (int i = 0; i < pos; ++i) power *= 10;

    return  num + ((long long) value - get_digit(num, pos)) * power;
}

int prefixSimilarity(long long num_a, long long num_b){
    int matched_digits = 0;
    for (int i = WIDTH - 1; i >= 0; --i)
        if (get_digit(num_a, i) == get_digit(num_b, i)) matched_digits++; else break;
```

```
    return matched_digits;
}

struct Node {
    vector<pair<int, Node*>> vs;
    Node* parent;
    long ping_time = INF;
    bool visited;
    long long num;
    int index;
};

void record_node(
        unordered_map<long long, Node*> &map,
        vector<int> &weights, vector<Node> &nodes,
        long long node_num, int node_ind) {

    vector<pair<int, Node*>> v;

    for (int i = 0; i < WIDTH; i++) {
        for (int d = 0; d < WIDTH; d++) {
            long long changed_num = set_digit(node_num, i, d);

            if (map.find(changed_num) != map.end())
                v.emplace_back(weights[prefixSimilarity(node_num, changed_num)],
(*map.find(changed_num)).second);
        }
    }

    for (int i = 0; i < WIDTH; i++) {
        for (int j = i + 1; j < WIDTH; j++) {

            long long swapped_num = set_digit(
                    set_digit(
                            node_num,
                            j,
                            get_digit(node_num, i)
                            ),
                    i,
                    get_digit(node_num, j)
                    );

            if(map.find(swapped_num) != map.end())
                v.emplace_back(weights[prefixSimilarity(node_num, swapped_num)],
(*map.find(swapped_num)).second);

        }
    }

    map[node_num] = &nodes[node_ind];

    for(pair<int, Node*> p : v){
        p.second->vs.emplace_back(p.first, &nodes[node_ind] );
        nodes[node_ind].vs.emplace_back(p.first, p.second );
    }
    nodes[node_ind].num = node_num;
    nodes[node_ind].index = node_ind;
}
```

```
void shortest_path(vector<Node> &nodes){
    priority_queue<pair<int, Node*>, vector<pair<int, Node*>>, greater<>> q;

    nodes[0].ping_time = 0;

    q.push( {0, &nodes[0]} );
    while(!q.empty()) {
        pair<int, Node*> p = q.top();
        q.pop();
        Node* node = p.second;

        if (node->visited) continue;
        else node->visited = true;

        for (pair<int, Node*> &v: node->vs) {
            Node* neighbour_node = v.second;
            int neighbour_cost = v.first;

            if (!neighbour_node->visited && neighbour_node->ping_time > node->ping_time +
neighbour_cost) {
                neighbour_node->parent = node;
                neighbour_node->ping_time = node->ping_time + neighbour_cost;
                q.push({neighbour_node->ping_time, neighbour_node});
            }
        }
    }
}

int main(){
    int n;
    cin >> n;

    vector<Node> nodes(n);

    vector<int> weights(WIDTH);
    for (int &weight: weights) cin >> weight;

    unordered_map<long long, Node*> map;

    for (int i = 0; i < n; i++) {
        long long num;
        cin >> num;
        record_node(map, weights, nodes, num, i);
    }

    vector<Node*> result;
    shortest_path(nodes);

    if (!nodes[n - 1].visited) {
        cout << -1;
    } else {
        cout << nodes[n - 1].ping_time << "\n";

        for (Node* node = &nodes[n - 1]; node; node = node->parent) {
            result.push_back(node);
        }

        cout << result.size() << "\n";

        for (int i = (int) result.size() - 1; i ⩾ 0; --i) {
```

```
            cout << 1 + result[i]->index << " ";
        }
    }
}
```