Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский университет ИТМО

МЕГАФАКУЛЬТЕТ ТРАНСЛЯЦИОННЫХ ИНФОРМАЦИОННЫЙ ТЕХНОЛОГИЙ

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРНАЯ РАБОТА №2

По дисциплине «Программирование»

Лабораторная работа. Анализ данных.

Выполнил Кудашев И.Э
(Фамилия Имя Отчество)

Проверил Повышев В.В
(Фамилия Имя Отчество)

Санкт-Петербург, 2021г

УСЛОВИЕ ЛАБОРАТОРНОЙ

На основании данных об остановках обещесвенного транспорта http://data.gov.spb.ru/opendata/7830001067-transport_station/ (можно скачать в xml)

Определить:

1. Маршрут с наибольшим количеством остановок по отдельными видам транспорта
2. Наиболее длинный маршрут (основывая на координатах) по отдельным видам транспорта
3. Улицу с наибольшим числом остановок

Для извлечения данных из xml-файла воспользоваться библиотекой pugixml, либо любым аналогом.

Для хранения и подсчета статистики, спроектировать и реализовать структуры данных, обеспечивающие оптимальную алгоритмическую сложность расчетов и не избыточность по памяти.

КОД

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstring>
#include <sstream>
#include <map>
#include <set>
#include <cmath>
#include ‘pugixml.hpp‘

using namespace std;
using std::pair;
using std::vector;
using std::cout;
using std::map;
using std::set;
using std::string;
using std::stringstream;

class XmlElem {
private:
    int num;
    string type_of_vehicle;
    string name_stopping;
    string the_official_name;
    vector<string> location;
    vector<string> routes;
    pair<double, double> coordinates;

public:
    XmlElem(int num, string type_of_vehicle, string name_stopping, string the_official_name, vector<string> location,
            vector<string> routes, pair<double, double> coordinates) :
            num(num), type_of_vehicle(type_of_vehicle), name_stopping(name_stopping),
            the_official_name(the_official_name), location(location), routes(routes), coordinates(coordinates) {};

    string get_type_of_vehicle() const {
        return type_of_vehicle;
    }

    string get_name_stopping() const {
        return name_stopping;
    }

    string get_the_official_name() const {
```

```
        return the_official_name;
    }

    vector<string> get_location() const {
        return location;
    }

    vector<string> get_routes() const {
        return routes;
    }

    double get_coordinates_x() const {
        return coordinates.first;
    }

    double get_coordinates_y() const {
        return coordinates.second;
    }

    int get_num() const {
        return num;
    }

    string get_index(int index) const {
        return routes[index];
    }

    int get_size() const {
        return routes.size();
    }


};

class Routes {
public:
    string route;
    vector<XmlElem> Bus;
    vector<XmlElem> Tram;
    vector<XmlElem> Trolleybus;
};

double coordinatesLength(const XmlElem &first, const XmlElem &second) {
    double first_latitude = first.get_coordinates_x() * M_PI / 180;
    double second_latitude = second.get_coordinates_x() * M_PI / 180;

    double first_longitude = first.get_coordinates_y() * M_PI / 180;
    double second_longitude = second.get_coordinates_y() * M_PI / 180;

    double general_latitude = second_latitude - first_latitude;
```

```
    double general_longitude = second_longitude - first_longitude;

    double answer = pow(sin(general_latitude / 2), 2) +
                    cos(first_latitude) * cos(second_latitude) * pow(sin(general_longitude / 2), 2);

    answer = 2 * asin(sqrt(answer)) * 6731; // 6731 - радиус земли

    return answer;
}

void separator(const string &source, string &firstItem, string &secondItem) {
    string div = ',';

    auto start = 0;
    auto end = source.find(div);

    while (end != -1) {
        firstItem = source.substr(start, end - start);
        start = end + div.length();
        end = source.find(div, start);
    }

    secondItem = source.substr(start, end);
}

string sepLocation(string &source) {
    vector<string| divisions{'ул.', ' ул.', ' УЛ.', ' ш.', ' Ш.', ' шоссе', ' ШОССЕ', ' пер.', ' ПЕР.', ' переулок',
                             ' ПЕРЕУЛОК', ' улица', 'улица', ' УЛИЦА', ' бул', ' БУЛ', ' бульвар', ' БУЛЬВАР',
                             ' пр',
                             ' ПР', ' проспект', ' ПРОСПЕКТ'};

    for (auto &division : divisions) {
        if (source.find(division) != -1) {
            int firstVal = source.find(division);
            int secondVal = firstVal + division.size();
            source.erase(firstVal, secondVal);
        }
    }

    if (source[source.size() - 1] == ' ') {
        source.erase(source.end() - 1);
    }
    return source;
}

void parser(vector<XmlElem| &source, map<string, Routes| &mappedRoutes, set<string| &nameRoutes,
            map<string, int| &locations) {
    pugi::xml_document document;
    document.load_file('data1.xml'); //подгружаем файл
```

```
pugi::xml_node data = document.child(‘dataset‘); //вытаскиваем корень

for (pugi::xml_node i = data.child(‘transport_station‘); i; i = i.next_sibling(‘transport_station‘)) {

    string firstParam;
    string secondParam;
    float firstValue;
    float secondValue;

    separator(i.child_value(‘coordinates‘), firstParam, secondParam);

    firstValue = stof(firstParam);
    secondValue = stof(secondParam);

    pair<double, double| coordinates;
    coordinates.first = firstValue;
    coordinates.second = secondValue;


    //ROUTES
    firstParam = ‘‘;
    secondParam = ‘‘;

    string strRoutes = i.child_value(‘routes‘), segment;
    stringstream tempStrRoutes(strRoutes);
    vector<string| vecRoutes;

    if (count(strRoutes.begin(), strRoutes.end(), ’,’)) {
        while (getline(tempStrRoutes, segment, ’,’)) {
            vecRoutes.push_back(segment);
        }
    } else {
        while (getline(tempStrRoutes, segment, ’.’)) {
            vecRoutes.push_back(segment);
        }
    }

    //LOCATIONS
    firstParam = ‘‘;
    secondParam = ‘‘;

    string strLocations = i.child_value(‘location‘);
    vector<string| vecLocations;
    stringstream tempStrLocations(strLocations);

    if (count(strLocations.begin(), strLocations.end(), ’,’) && !strLocations.empty()) {
        while (getline(tempStrLocations, segment, ’,’)) {
            if (segment[0] == ’ ’) {
```

```
                segment.erase(segment.begin());
            }
            vecLocations.push_back(sepLocation(segment));
            locations[sepLocation(segment)] += 1;
        }
    } else if (!strLocations.empty()) {
        vecLocations.push_back(sepLocation(strLocations));
        locations[sepLocation(strLocations)] += 1;
    }

    int numValue = stoi(i.child_value("number"));
    string type_of_vehicle = i.child_value("type_of_vehicle");
    string name_stopping = i.child_value("name_stopping");
    string the_official_name = i.child_value("the_official_name");
    source.emplace_back(
        XmlElem(numValue, type_of_vehicle, name_stopping, the_official_name, vecLocations, vecRoutes,
            coordinates));

    if (!strcmp(i.child_value("type_of_vehicle"), "Трамвай")) {
        for (int j = 0; j < vecRoutes.size(); ++j) {
            mappedRoutes[vecRoutes[j]].Tram.emplace_back(
                XmlElem(numValue, type_of_vehicle, name_stopping, the_official_name, vecLocations, vecRoutes,
                    coordinates));

            mappedRoutes[vecRoutes[j]].route = vecRoutes[j];
            nameRoutes.insert(vecRoutes[j]);
        }
    } else if (!strcmp(i.child_value("type_of_vehicle"), "Автобус")) {
        for (int j = 0; j < vecRoutes.size(); ++j) {
            mappedRoutes[vecRoutes[j]].Bus.emplace_back(
                XmlElem(numValue, type_of_vehicle, name_stopping, the_official_name, vecLocations, vecRoutes,
                    coordinates));

            mappedRoutes[vecRoutes[j]].route = vecRoutes[j];
            nameRoutes.insert(vecRoutes[j]);
        }
    } else if (!strcmp(i.child_value("type_of_vehicle"), "Троллейбус")) {
        for (int j = 0; j < vecRoutes.size(); ++j) {
            mappedRoutes[vecRoutes[j]].Trolleybus.emplace_back(
                XmlElem(numValue, type_of_vehicle, name_stopping, the_official_name, vecLocations, vecRoutes,
                    coordinates));

            mappedRoutes[vecRoutes[j]].route = vecRoutes[j];
            nameRoutes.insert(vecRoutes[j]);
        }
    }
  }
}
```

```
void set_all_information(vector<XmlElem| &element, map<string, int| &tram_routes, map<string, int| &bus_routes,
                         map<string, int| &trolleybus_routes) {
    for (auto const &i : element) {
        if (i.get_type_of_vehicle() == 'Трамвай') {
            for (int j = 0; j < i.get_size(); j++)
                tram_routes[i.get_index(j)] += 1;
        } else if (i.get_type_of_vehicle() == 'Автобус') {
            for (int j = 0; j < i.get_size(); j++)
                bus_routes[i.get_index(j)] += 1;
        } else if (i.get_type_of_vehicle() == 'Троллейбус') {
            for (int j = 0; j < i.get_size(); j++)
                trolleybus_routes[i.get_index(j)] += 1;
        }
    }
}

void set_counter(map<string, int| &routes, int &counter, string &max_routes) {
    for (auto const i : routes)
        if (i.second | counter) {
            counter = i.second;
            max_routes = i.first;
        }
}


void set_routes(set<string| &names, map<string, Routes| &routes, map<string, float| &tram_routes_size,
                map<string, float| &bus_routes_size, map<string, float| &trolleybus_routes_size) {
    for (auto const j : names) {
        if (routes[j].Tram.size() | 1) {
            for (int k = 0; k < routes[j].Tram.size() - 1; k++)
                tram_routes_size[routes[j].route] += coordinatesLength(routes[j].Tram[k], routes[j].Tram[k + 1]);
        }
        if (routes[j].Bus.size() | 1) {
            for (int k = 0; k < routes[j].Bus.size() - 1; k++)
                bus_routes_size[routes[j].route] += coordinatesLength(routes[j].Bus[k], routes[j].Bus[k + 1]);
        }
        if (routes[j].Trolleybus.size() | 1) {
            for (int k = 0; k < routes[j].Trolleybus.size() - 1; k++)
                trolleybus_routes_size[routes[j].route] += coordinatesLength(routes[j].Trolleybus[k],
                                                                             routes[j].Trolleybus[k + 1]);
        }
    }
}

void set_size(map<string, float| &routes_size, float &path_counter, string &max_route) {
    for (auto const &i : routes_size) {
        if (i.second | path_counter) {
            path_counter = i.second;
            max_route = i.first;
        }
```

```
    }
}

void max_routes(int &maximum, string &max_size, map<string, int| &locations) {
    for (auto const i : locations) {
        if (i.second | maximum) {
            maximum = i.second;
            max_size = i.first;
        }
    }
}

int main() {
    vector<XmlElem| stations;
    map<string, Routes| routes;
    set<string| names;

    map<string, int| locations;
    map<string, int| tram_routes, bus_routes, trolleybus_routes;
    map<string, float| tram_routes_size, bus_routes_size, trolleybus_routes_size;


    string max_tram_routes, max_bus_routes, max_trolleybus_routes;
    string max_tram_route, max_bus_route, max_trolleybus_route;
    int tram_counter = 0, bus_counter = 0, trolleybus_counter = 0;
    float tram_path_counter = 0, bus_path_counter = 0, trolleybus_path_counter = 0;
    int maximum = 0;
    string maximum_size;

    parser(stations, routes, names, locations);
    set_all_information(stations, tram_routes, bus_routes, trolleybus_routes);

    set_counter(tram_routes, tram_counter, max_tram_routes);
    set_counter(bus_routes, bus_counter, max_bus_routes);
    set_counter(tram_routes, trolleybus_counter, max_trolleybus_routes);

    cout << 'Максимальное количество остановок трамвая равно: ' << tram_counter << ' - № маршрута -  '
         << max_trolleybus_routes << endl;
    cout << 'Максимальное количество остановок автобуса равно: ' << bus_counter << ' - № маршрута - ' <<
max_bus_routes
         << endl;
    cout << 'Максимальное количество остановок трамвая равно: ' << trolleybus_counter << '- № маршрута -  '
         << max_trolleybus_routes << endl;

    set_routes(names, routes, tram_routes_size, bus_routes_size, trolleybus_routes_size);

    set_size(tram_routes_size, tram_path_counter, max_tram_route);
    set_size(bus_routes_size, bus_path_counter, max_bus_route);
    set_size(trolleybus_routes_size, trolleybus_path_counter, max_trolleybus_route);
```

```
    cout << '\n';
    cout << 'Наиболее длиный маршрут трамвая: ' << max_tram_route << endl;
    cout << 'Наиболее длиный маршрут автобуса: ' << max_bus_route << endl;
    cout << 'Наиболее длиный маршрут троллейбуса: ' << max_trolleybus_route << endl;

    max_routes(maximum, maximum_size, locations);
    cout << '\n' << 'Максимальное количество на улице ' << maximum_size << ' = ' << maximum << endl;

    return 0;
}
```

Кудашев И.Э