**ЛАБОРАТОРНАЯ РАБОТА №3**

**Цель работы:** Сравнить массив, связанный список и стандарт библиотеки языка.

**Задание:** В городе расположены n автобусных остановок, обозначенных числами из N={1,2,....,n}. Имеется R автобусных маршрутов, заданных последовательностями соседних остановок при движении автобуса в одном направлении:

М1={I11,I12,...,I1m1},

М2={I21,I22,...,I2m2},

.....

Mr={Ir1,Ir2,...,Irmr},

где Ijk натуральное.

Написать программу, которая по заданным номерам остановок I и J определяет наиболее быстрый путь перемещения пассажира из остановки I в остановку J с использованием имеющихся маршрутов автобусов при условий, что время движения между соседними остановками у всех маршрутов одинаково и в 3 раза меньше времени изменения маршрута. Кроме того, автобусы могут двигаться в обоих направлениях.

Каждая задача требует использовать некоторую структуру данных : стек, очередь и тд

Следует реализовать структуру данных 3 способами

А) через массив

Б) через связанный список

В) с использованием стандартной библиотеки языка (например, STL для С++)

Сравнить работоспособность и производительность каждой реализации.

**Описание работы программы:**

Вводиться количество остановок, количество маршрутов. Программа создаёт маршруты, вводится количество останов для маршрута, а после сами остановки. Вводятся начальная остановка и конечная остановка и программа начинает свою работу. Программа находит кротчайший маршрут из начальной остановки до конечная остановки и выводит результат.

**Листинг программы через простой массив:**

##include <iostream>

#include <queue>

#include <limits>

#include <chrono>

using namespace std;

const int INF = numeric\_limits<int>::max();

const int TRANSFER\_TIME = 3;

const int MAX\_STOPS = 100; //Максимальное количество остановок

const int MAX\_ROUTES = 50; //Максимальное количество маршрутов

struct State {

int stop;

int route\_index;

int dist;

int transfers;

};

bool compareStates(const State& a, const State& b) {

if (a.dist != b.dist) {

return a.dist > b.dist;

}

return a.transfers > b.transfers;

}

int main() {

system("chcp 1251");

system("cls");

cout << "Выполнил: Карпенко Денис Иванович\nГруппа: 020303-АИСа-24о\n";

int n, r;

cout << "Введите количество остановок (n): ";

cin >> n;

cout << "Введите количество маршрутов (r): ";

if (n > MAX\_STOPS) {

cout << "Превышено максимальное количество остановок (" << MAX\_STOPS << ")" << endl;

return 1;

}

cin >> r;

if (r > MAX\_ROUTES) {

cout << "Превышено максимальное количество маршрутов (" << MAX\_ROUTES << ")" << endl;

return 1;

}

int routes[MAX\_ROUTES][MAX\_STOPS];

int route\_lengths[MAX\_ROUTES];

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

cout << "Введите длину маршрута " << i + 1 << ": ";

cin >> route\_lengths[i];

if (route\_lengths[i] > MAX\_STOPS) {

cout << "Превышена максимальная длина маршрута (" << MAX\_STOPS << ")" << endl;

return 1;

}

cout << "Введите остановки для маршрута " << i + 1 << ": ";

for (int j = 0; j < route\_lengths[i]; ++j) {

cin >> routes[i][j];

routes[i][j]--;

}

}

int start\_stop, end\_stop;

cout << "Введите начальную остановку (I): ";

cin >> start\_stop;

start\_stop--;

cout << "Введите конечную остановку (J): ";

cin >> end\_stop;

end\_stop--;

auto start = chrono::high\_resolution\_clock::now();

int min\_dist[MAX\_STOPS][MAX\_ROUTES];

int min\_transfers[MAX\_STOPS][MAX\_ROUTES];

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

for (int j = 0; j < r; ++j) {

min\_dist[i][j] = INF;

min\_transfers[i][j] = INF;

}

}

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

for (int j = 0; j < route\_lengths[i]; ++j) {

if (routes[i][j] == start\_stop) {

min\_dist[start\_stop][i] = 0;

min\_transfers[start\_stop][i] = 0;

break;

}

}

}

State pq[MAX\_STOPS \* MAX\_ROUTES];

int pq\_size = 0;

auto push\_pq = [&](const State& s) {

if (pq\_size < MAX\_STOPS \* MAX\_ROUTES) {

pq[pq\_size] = s;

int i = pq\_size;

while (i > 0 && compareStates(pq[i], pq[(i - 1) / 2])) {

swap(pq[i], pq[(i - 1) / 2]);

i = (i - 1) / 2;

}

pq\_size++;

}

else {

cout << "Ошибка: переполнение очереди приоритетов" << endl;

}//проверка на переполнение

};

auto pop\_pq = [&]() -> void {

if (pq\_size == 0) return;

pq[0] = pq[pq\_size - 1];

pq\_size--;

int i = 0;

while (true) {

int left = 2 \* i + 1;

int right = 2 \* i + 2;

int smallest = i;

if (left < pq\_size && compareStates(pq[left], pq[smallest])) {

smallest = left;

}

if (right < pq\_size && compareStates(pq[right], pq[smallest])) {

smallest = right;

}

if (smallest == i) break;

swap(pq[i], pq[smallest]);

i = smallest;

}

};

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

for (int j = 0; j < route\_lengths[i]; ++j) {

if (routes[i][j] == start\_stop) {

push\_pq({ start\_stop, i, 0, 0 });

break;

}

}

}

while (pq\_size > 0) {

State current\_state = pq[0];

pop\_pq();

int u = current\_state.stop;

int current\_route\_index = current\_state.route\_index;

int d = current\_state.dist;

int t = current\_state.transfers;

if (d > min\_dist[u][current\_route\_index] || t > min\_transfers[u][current\_route\_index]) continue;

for (int u\_index = 0; u\_index < route\_lengths[current\_route\_index]; ++u\_index) {

if (routes[current\_route\_index][u\_index] == u) {

if (u\_index + 1 < route\_lengths[current\_route\_index]) {

int v = routes[current\_route\_index][u\_index + 1];

if (min\_dist[v][current\_route\_index] > d + 1) {

min\_dist[v][current\_route\_index] = d + 1;

min\_transfers[v][current\_route\_index] = t;

push\_pq({ v, current\_route\_index, d + 1, t });

}

else if (min\_dist[v][current\_route\_index] == d + 1 && min\_transfers[v][current\_route\_index] > t) {

min\_transfers[v][current\_route\_index] = t;

push\_pq({ v, current\_route\_index, d + 1, t });

}

}

if (u\_index - 1 >= 0) {

int v = routes[current\_route\_index][u\_index - 1];

if (min\_dist[v][current\_route\_index] > d + 1) {

min\_dist[v][current\_route\_index] = d + 1;

min\_transfers[v][current\_route\_index] = t;

push\_pq({ v, current\_route\_index, d + 1, t });

}

else if (min\_dist[v][current\_route\_index] == d + 1 && min\_transfers[v][current\_route\_index] > t) {

min\_transfers[v][current\_route\_index] = t;

push\_pq({ v, current\_route\_index, d + 1, t });

}

}

break;

}

}

//пересадка

for (int other\_route\_index = 0; other\_route\_index < r; ++other\_route\_index) {

if (other\_route\_index != current\_route\_index) {

for (int j = 0; j < route\_lengths[other\_route\_index]; ++j) {

int v = routes[other\_route\_index][j];

if (v == u) {

if (min\_dist[u][other\_route\_index] > d + TRANSFER\_TIME) {

min\_dist[u][other\_route\_index] = d + TRANSFER\_TIME;

min\_transfers[u][other\_route\_index] = t + 1;

push\_pq({ u, other\_route\_index, d + TRANSFER\_TIME, t + 1 });

}

else if (min\_dist[u][other\_route\_index] == d + TRANSFER\_TIME && min\_transfers[u][other\_route\_index] > t + 1) {

min\_transfers[u][other\_route\_index] = t + 1;

push\_pq({ u, other\_route\_index, d + TRANSFER\_TIME, t + 1 });

}

break;

}

}

}

}

}

int best\_dist = INF;

int best\_transfers = INF;

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

best\_dist = min(best\_dist, min\_dist[end\_stop][i]);

}

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

if (min\_dist[end\_stop][i] == best\_dist) {

best\_transfers = min(best\_transfers, min\_transfers[end\_stop][i]);

}

}

if (best\_dist == INF) {

cout << "Нет пути от остановки " << start\_stop + 1 << " до остановки " << end\_stop + 1 << endl;

}

else {

cout << "Наименьшее время от остановки " << start\_stop + 1 << " до остановки " << end\_stop + 1 << ": " << best\_dist << endl;

//cout << "Количество пересадок: " << best\_transfers << endl;

}

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> elapsed\_seconds = end - start;

double time = elapsed\_seconds.count();

cout << "Время выполнения: " << time << " секунд" << endl;

return 0;

}

**Листинг программы через связанный список:**

#include <iostream>

#include <vector>

#include <queue>

#include <limits>

#include <chrono>

using namespace std;

const int INF = numeric\_limits<int>::max();

const int TRANSFER\_TIME = 3;

struct StopNode {

int stop;

StopNode\* next;

StopNode\* prev;

StopNode(int stop\_num) : stop(stop\_num), next(nullptr), prev(nullptr) {}

};

struct State {

int stop; //номер остановки

int route\_index; //индекс маршрута, на котором находимся

int dist; //расстояние от н ост

int transfers; //кол пересадок

bool operator>(const State& other) const {

if (dist != other.dist) {

return dist > other.dist;

}

return transfers > other.transfers;

}

};

int main() {

system("chcp 1251");

system("cls");

cout << "Выполнил: Карпенко Денис Иванович\nГруппа: 020303-АИСа-24о\n";

int n, r;

cout << "Введите количество остановок (n): ";

cin >> n;

cout << "Введите количество маршрутов (r): ";

cin >> r;

vector<StopNode\*> routes(r, nullptr); //массив указателей на начало каждого маршрута

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

int route\_length;

cout << "Введите длину маршрута " << i + 1 << ": ";

cin >> route\_length;

StopNode\* head = nullptr;

StopNode\* tail = nullptr;

cout << "Введите остановки для маршрута " << i + 1 << ": ";

for (int j = 0; j < route\_length; ++j) {

int stop\_num;

cin >> stop\_num;

stop\_num--;

StopNode\* newNode = new StopNode(stop\_num);

if (!head) {

head = newNode;

tail = newNode;

}

else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

routes[i] = head; //указатель на начало маршрута

}

int start\_stop, end\_stop;

cout << "Введите начальную остановку (I): ";

cin >> start\_stop;

start\_stop--;

cout << "Введите конечную остановку (J): ";

cin >> end\_stop;

end\_stop--;

vector<vector<int>> min\_dist(n, vector<int>(r, INF));

vector<vector<int>> min\_transfers(n, vector<int>(r, INF));

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

StopNode\* current = routes[i];

while (current) {

if (current->stop == start\_stop) {

min\_dist[start\_stop][i] = 0;

min\_transfers[start\_stop][i] = 0;

break;

}

current = current->next;

}

}

auto start = chrono::high\_resolution\_clock::now();

priority\_queue<State, vector<State>, greater<State>> pq;

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

StopNode\* current = routes[i];

while (current) {

if (current->stop == start\_stop) {

pq.push({ start\_stop, i, 0, 0 });

break;

}

current = current->next;

}

}

while (!pq.empty()) {

State current\_state = pq.top();

pq.pop();

int u = current\_state.stop;

int route\_index = current\_state.route\_index;

int d = current\_state.dist;

int t = current\_state.transfers;

if (d > min\_dist[u][route\_index] || t > min\_transfers[u][route\_index]) continue;

StopNode\* current\_node = routes[route\_index];

while (current\_node && current\_node->stop != u) {

current\_node = current\_node->next;

}

if (current\_node) {

if (current\_node->next) {

int v = current\_node->next->stop;

if (min\_dist[v][route\_index] > d + 1) {

min\_dist[v][route\_index] = d + 1;

min\_transfers[v][route\_index] = t;

pq.push({ v, route\_index, d + 1, t });

}

else if (min\_dist[v][route\_index] == d + 1 && min\_transfers[v][route\_index] > t) {

min\_transfers[v][route\_index] = t;

pq.push({ v, route\_index, d + 1, t });

}

}

if (current\_node->prev) {

int v = current\_node->prev->stop;

if (min\_dist[v][route\_index] > d + 1) {

min\_dist[v][route\_index] = d + 1;

min\_transfers[v][route\_index] = t;

pq.push({ v, route\_index, d + 1, t });

}

else if (min\_dist[v][route\_index] == d + 1 && min\_transfers[v][route\_index] > t) {

min\_transfers[v][route\_index] = t;

pq.push({ v, route\_index, d + 1, t });

}

}

//пересадка

for (int other\_route\_index = 0; other\_route\_index < r; ++other\_route\_index) {

if (other\_route\_index != route\_index) {

StopNode\* other\_route\_node = routes[other\_route\_index];

bool can\_transfer = false;

while (other\_route\_node) {

if (other\_route\_node->stop == u) {

can\_transfer = true;

break;

}

other\_route\_node = other\_route\_node->next;

}

if (can\_transfer) {

if (min\_dist[u][other\_route\_index] > d + TRANSFER\_TIME) {

min\_dist[u][other\_route\_index] = d + TRANSFER\_TIME;

min\_transfers[u][other\_route\_index] = t + 1;

pq.push({ u, other\_route\_index, d + TRANSFER\_TIME, t + 1 });

}

else if (min\_dist[u][other\_route\_index] == d + TRANSFER\_TIME && min\_transfers[u][other\_route\_index] > t + 1) {

min\_transfers[u][other\_route\_index] = t + 1;

pq.push({ u, other\_route\_index, d + TRANSFER\_TIME, t + 1 });

}

}

}

}

}

}

int best\_dist = INF;

int best\_transfers = INF;

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

best\_dist = min(best\_dist, min\_dist[end\_stop][i]);

}

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

if (min\_dist[end\_stop][i] == best\_dist) {

best\_transfers = min(best\_transfers, min\_transfers[end\_stop][i]);

}

}

if (best\_dist == INF) {

cout << "Нет пути от остановки " << start\_stop + 1 << " до остановки " << end\_stop + 1 << endl;

}

else {

cout << "Наименьшее время от остановки " << start\_stop + 1 << " до остановки " << end\_stop + 1 << ": " << best\_dist << endl;

//cout << "Количество пересадок: " << best\_transfers << endl;

}

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

StopNode\* current = routes[i];

while (current) {

StopNode\* next = current->next;

delete current;

current = next;

}

}

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> elapsed\_seconds = end - start;

double time = elapsed\_seconds.count();

cout << "Время выполнения: " << time << " секунд" << endl;

return 0;

}

**Листинг программы через использование стандартной библиотеки языка:**

#include <iostream>

#include <vector>

#include <queue>

#include <limits>

#include <algorithm>

#include <chrono>

using namespace std;

const int INF = numeric\_limits<int>::max();

const int TRANSFER\_TIME = 3;

struct State {

int stop;

int route\_index;

int dist;

int transfers;

bool operator>(const State& other) const {

if (dist != other.dist) {

return dist > other.dist;

}

return transfers > other.transfers;

}

};

int main() {

system("chcp 1251");

system("cls");

cout << "Выполнил: Карпенко Денис Иванович\nГруппа: 020303-АИСа-24о\n";

int n, r;

cout << "Введите количество остановок (n): ";

cin >> n;

cout << "Введите количество маршрутов (r): ";

cin >> r;

vector<vector<int>> routes(r);

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

int route\_length;

cout << "Введите длину маршрута " << i + 1 << ": ";

cin >> route\_length;

routes[i].resize(route\_length);

cout << "Введите остановки для маршрута " << i + 1 << ": ";

for (int j = 0; j < route\_length; ++j) {

cin >> routes[i][j];

routes[i][j]--; // -1, чтоб исп индек

}

}

int start\_stop, end\_stop;

cout << "Введите начальную остановку (I): ";

cin >> start\_stop;

start\_stop--;

cout << "Введите конечную остановку (J): ";

cin >> end\_stop;

end\_stop--;

auto start = chrono::high\_resolution\_clock::now();

vector<vector<int>> min\_dist(n, vector<int>(r, INF));

vector<vector<int>> min\_transfers(n, vector<int>(r, INF));

//иниц начальной остановки на всех маршрутах, где она есть

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

auto it = find(routes[i].begin(), routes[i].end(), start\_stop);

if (it != routes[i].end()) {

min\_dist[start\_stop][i] = 0;

min\_transfers[start\_stop][i] = 0;

}

}

priority\_queue<State, vector<State>, greater<State>> pq;

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

auto it = find(routes[i].begin(), routes[i].end(), start\_stop);

if (it != routes[i].end()) {

pq.push({ start\_stop, i, 0, 0 }); //Добавляем начальную остановку на каждый маршрут, где она есть

}

}

while (!pq.empty()) {

State current = pq.top();

pq.pop();

int u = current.stop;

int current\_route\_index = current.route\_index;

int d = current.dist;

int t = current.transfers;

if (d > min\_dist[u][current\_route\_index] || t > min\_transfers[u][current\_route\_index]) continue;

auto& current\_route = routes[current\_route\_index];

auto it = find(current\_route.begin(), current\_route.end(), u);

if (it != current\_route.end()) {

int u\_index = distance(current\_route.begin(), it);

//движение вперед по маршруту

if (u\_index + 1 < current\_route.size()) {

int v = current\_route[u\_index + 1];

if (min\_dist[v][current\_route\_index] > d + 1) {

min\_dist[v][current\_route\_index] = d + 1;

min\_transfers[v][current\_route\_index] = t;

pq.push({ v, current\_route\_index, d + 1, t });

}

else if (min\_dist[v][current\_route\_index] == d + 1 && min\_transfers[v][current\_route\_index] > t) {

min\_transfers[v][current\_route\_index] = t;

pq.push({ v, current\_route\_index, d + 1, t });

}

}

//движение назад по маршруту

if (u\_index - 1 >= 0) {

int v = current\_route[u\_index - 1];

if (min\_dist[v][current\_route\_index] > d + 1) {

min\_dist[v][current\_route\_index] = d + 1;

min\_transfers[v][current\_route\_index] = t;

pq.push({ v, current\_route\_index, d + 1, t });

}

else if (min\_dist[v][current\_route\_index] == d + 1 && min\_transfers[v][current\_route\_index] > t) {

min\_transfers[v][current\_route\_index] = t;

pq.push({ v, current\_route\_index, d + 1, t });

}

}

//пересадки

for (int other\_route\_index = 0; other\_route\_index < r; ++other\_route\_index) {

if (other\_route\_index != current\_route\_index) {

auto it2 = find(routes[other\_route\_index].begin(), routes[other\_route\_index].end(), u);

if (it2 != routes[other\_route\_index].end()) {

if (min\_dist[u][other\_route\_index] > d + TRANSFER\_TIME) {

min\_dist[u][other\_route\_index] = d + TRANSFER\_TIME;

min\_transfers[u][other\_route\_index] = t + 1;

pq.push({ u, other\_route\_index, d + TRANSFER\_TIME, t + 1 });

}

else if (min\_dist[u][other\_route\_index] == d + TRANSFER\_TIME && min\_transfers[u][other\_route\_index] > t + 1) {

min\_transfers[u][other\_route\_index] = t + 1;

pq.push({ u, other\_route\_index, d + TRANSFER\_TIME, t + 1 });

}

}

}

}

}

}

int dist = INF;

int transfers = INF;

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

dist = min(dist, min\_dist[end\_stop][i]);

}

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

if (min\_dist[end\_stop][i] == dist) {

transfers = min(transfers, min\_transfers[end\_stop][i]);

}

}

if (dist == INF) {

cout << "Нет пути от остановки " << start\_stop + 1 << " до остановки " << end\_stop + 1 << endl;

}

else {

cout << "Наименьшее время от остановки " << start\_stop + 1 << " до остановки " << end\_stop + 1 << ": " << dist << endl;

//cout << "Количество пересадок: " << transfers << endl;

}

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> elapsed\_seconds = end - start;

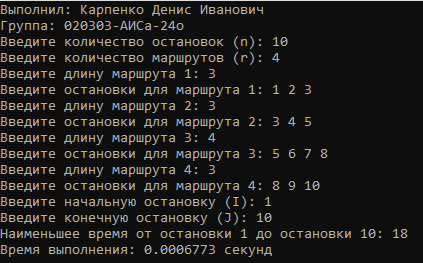
double time = elapsed\_seconds.count();

cout << "Время выполнения: " << time << " секунд" << endl;

return 0;

}

**Пример работы программы:**



**Вывод:** Все три вида хранения данных показывали разные результаты, по которым можно делать выводы что программа сделанная через массив самая быстрая, но пр и этом самая сложная, так надо не забывать правильно освобождать память. Программа через связанный список получилась средней по скорости или на равне с программой с использованием стандартной библиотеки языка. Использование стандартной библиотеки языка показало самый медленный результат, но не сильно уступая другим программам. Такая программа самая простая в написании и самая эффективная.