Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №8

по курсу “Логика и основа алгоритмизации в ИЗ”

на тему “Обход графа в ширину”



Пенза 2022

**Название**

Определение характеристик графов.

**Цель работы:** выполнить лабораторные указания 1-2 выполняя обход графа в ширину.

**Лабораторное задание:**

**Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) матрицу смежности для неориентированного графа *G*. Выведите матрицу на экран.
2. Для сгенерированного графа осуществите процедуру обхода в ширину, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди используйте класс **queue** из стандартной библиотеки С++.

**3.** Реализуйте процедуру обхода в ширину для графа, представленного списками смежности.

**Задание 2**

1. Для матричной формы представления графов реализуйте алгоритм обхода в ширину с использованием очереди, построенной на основе структуры данных «список», самостоятельно созданной в лабораторной работе № 3.
2. Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

**Задание 1:**

**Листинг**

**//L\_ptr.cpp**

**#include "L\_ptr.h"**

**template<typename T>**

**void L\_ptr<T>::Display() {**

**List<T>\* x = head;**

**while (x != nullptr) {**

**std::cout << x->number << std::endl;**

**x = x->next;**

**}**

**}**

**template<typename T>**

**int L\_ptr<T>::at(int count) {**

**if (this->size == 0 || count < 0 || count > size) return -1;**

**else {**

**return this->operator [](count);**

**}**

**}**

**template<typename T>**

**List<T>\* L\_ptr<T>::\_\_index\_\_(int count) {**

**if (count < 0 || count >= size) nullptr;**

**int id = 0;**

**List<T>\* tmp = head;**

**while (id + 1 != count) {**

**tmp = tmp->next;**

**id++;**

**}**

**return tmp;**

**}**

**template<typename T>**

**void L\_ptr<T>::Del\_vertex(int count) {**

**size--;**

**if (count == 0) {**

**List<T>\* tmp = head;**

**head = head->next;**

**tmp = nullptr;**

**}**

**else {**

**List<T>\* tmp = \_\_index\_\_(count);**

**tmp->next = tmp->next->next;**

**}**

**};**

**template<typename T>**

**int L\_ptr<T>::lenght() { return size; };**

**template<typename T>**

**List<T>\* top() {**

**if (this->size <= 0) {**

**std::cout << "Данных нет" << std::endl;**

**\_sleep(3000);**

**abort();**

**}**

**return this->head;**

**};**

**template<typename T>**

**void L\_ptr<T>::pop() {**

**if (this->size <= 0) {**

**std::cout << "Данных нет" << std::endl;**

**}**

**List<T>\* tmp = this->head;**

**this->head = this->head->next;**

**tmp->next = nullptr;**

**this->size--;**

**delete tmp;**

**};**

**template<typename T>**

**List<T>\* L\_ptr<T>::find(T data) {**

**if (size == 0) {**

**abort();**

**}**

**List<T>\* tmp = this->head;**

**while (tmp->number != data) {**

**if (tmp->next == nullptr) {**

**return this->head;**

**}**

**tmp = tmp->next;**

**}**

**return tmp;**

**}**

**template<typename T>**

**int L\_ptr<T>::operator [] (int a) {**

**List<T>\* x = head;**

**if (size == 0) {**

**std::cout << "Лист пуст" << std::endl;**

**return 0;**

**}**

**while (a != 0) {**

**x = x->next;**

**a--;**

**if (a != 0 && x == nullptr || a != 0 && x->next == nullptr || a == 0 && x == nullptr) {**

**std::cout << "Ошибка скобок" << std::endl;**

**return 0;**

**}**

**}**

**return x->number;**

**}**

**template<typename T>**

**void L\_ptr<T>::Swap(int top1, int top2) {**

**List<T>\* head\_tmp1 = nullptr;**

**List<T>\* head\_tmp2 = nullptr;**

**List<T>\* tmp1 = nullptr, \* tmp2 = nullptr;**

**List<T>\* x = head;**

**if (top1 == 0) {**

**tmp1 = head;**

**head\_tmp1 = head;**

**}**

**if (top2 >= size) {**

**std::cout << "Ошибка скобок" << std::endl;**

**return;**

**}**

**while (top2 != 0) {**

**if (top1 == 1) { head\_tmp1 = x; tmp1 = x->next; }**

**if (top2 == 1) { head\_tmp2 = x; tmp2 = x->next; }**

**x = x->next;**

**top1--; top2--;**

**}**

**int g;**

**tmp2 = tmp1;**

**tmp1 = x;**

**//**

**g = tmp2->number;**

**tmp2->number = tmp1->number;**

**tmp1->number = g;**

**}**

**//**

**//**

**// L\_ptr.h**

**#pragma once**

**// L\_ptr.h**

**#include "List.cpp"**

**#include <iostream>**

**template <typename T>**

**class L\_ptr {**

**protected:**

**int size;**

**List<T>\* head = nullptr;**

**List<T>\* end = nullptr;**

**List<T>\* \_\_index\_\_(int count);**

**public:**

**L\_ptr() : size(0) { };**

**L\_ptr(int size) : size(size) { };**

**int operator [] (int a);**

**const List<T>\* Get\_head() { return head; };**

**List<T>\* top();**

**void pop();**

**// находим элемент по данным**

**List<T>\* find(T data);**

**// удаление узла**

**void Del\_vertex(int count);**

**// меняем местами два узла**

**void Swap(int top1, int top2);**

**// проход по списку**

**int at(int count);**

**// получить длину списка**

**int lenght();**

**// вывести все на экран**

**void Display();**

**};**

**//**

**//**

**// List.cpp**

**template <typename T>**

**class L\_ptr;**

**template <typename T>**

**class List {**

**public:**

**List\* next = nullptr;**

**T number;**

**// конструктор**

**List() : number(0), next(nullptr) { };**

**List(int number) : number(number), next(nullptr) { };**

**List(const List& a) {**

**this->number = a.number;**

**this->next = a.next;**

**};**

**// булевые операторы**

**bool operator == (List& a) { return number == a.number; };**

**bool operator >= (List& a) { return number >= a.number; };**

**bool operator <= (List& a) { return number <= a.number; };**

**bool operator < (List& a) { return number < a.number; };**

**bool operator > (List& a) { return number > a.number; };**

**bool operator == (int a) { return number == a; };**

**bool operator >= (int a) { return number >= a; };**

**bool operator <= (int a) { return number <= a; };**

**bool operator < (int a) { return number < a; };**

**bool operator > (int a) { return number > a; };**

**//**

**T& operator = (List\* a) { //**

**this->number = a->number;**

**return this->number;**

**};**

**operator int() { return this->number; };**

**~List() { delete next; next = nullptr; }**

**friend L\_ptr<T>;**

**};**

**//**

**//**

**// laba\_8.cpp**

**#include <iostream>**

**#include <locale>**

**#include <queue>**

**#include "L\_ptr.cpp"**

**using namespace std;**

**template <typename T>**

**class queuee : public L\_ptr<T> {**

**public:**

**queuee() : L\_ptr<T>() { };**

**void Push(List<T>\* g) {**

**List<T>\* tmp = new List<T>;**

**\*tmp = g;**

**if (this->size == 0) {**

**this->end = tmp;**

**this->head = tmp;**

**}**

**else {**

**this->end->next = tmp;**

**this->end = tmp;**

**}**

**this->size++;**

**};**

**};**

**int main()**

**{**

**setlocale(LC\_ALL, "RUS");**

**srand(time(NULL));**

**/\***

**1. Сгенерируйте (используя генератор случайных чисел) матрицу**

**смежности для неориентированного графа G. Выведите матрицу на экран.**

**\*/**

**int count;**

**cout << "Введите размер матрицы" << endl;**

**cin >> count;**

**cout << endl << endl;**

**cout << "Матрица смежности" << endl;**

**int\*\* arr = new int\* [count];**

**for (int i = 0; i != count; i++) {**

**arr[i] = new int[count];**

**for (int j = 0; j != count; j++) {**

**arr[i][j] = 0;**

**}**

**}**

**for (int i = 0; i < count; i++) {**

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

**int r = rand() % 101;**

**if (r > 40) {**

**arr[i][j] = 1;**

**arr[j][i] = 1;**

**}**

**else {**

**arr[i][j] = 0;**

**arr[j][i] = 0;**

**}**

**arr[i][i] = 0;**

**}**

**}**

**for (int i = 0; i != count; i++) {**

**for (int j = 0; j != count; j++) {**

**cout << arr[i][j] << ' ';**

**}**

**cout << endl;**

**}**

**cout << endl << endl;**

**/\***

**2. Для сгенерированного графа осуществите**

**процедуру обхода в ширину**

**\*/**

**clock\_t start, end; // объявляем переменные для определения времени выполнения**

**queue<int> Queue;**

**int\* visited = new int[count];**

**for (int i = 0; i != count; i++) {**

**start = clock();**

**for (int i = 0; i != count; i++) visited[i] = 0;**

**Queue.push(i);**

**cout << "Вершина " << i << endl;**

**while (!Queue.empty()) {**

**int node = Queue.front();**

**Queue.pop();**

**if (visited[node] == 1) continue;**

**visited[node] = 1;**

**for (int i = 0; i != count; i++) {**

**if (arr[node][i] == 0 || visited[i] == 1) continue;**

**Queue.push(i);**

**}**

**cout << node << ' ';**

**}**

**cout << endl;**

**double time = 0.0;**

**end = clock();**

**time += (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("Время работы: %lf в секундах\n\n", time);**

**}**

**/\***

**3. Реализуйте процедуру обхода в ширину для**

**графа, представленного списками смежности.**

**\*/**

**int\*\* arr2 = new int\*[count];**

**for (int i = 0; i < count; i++) {**

**int g = 1;**

**for (int j = 0; j < count; j++) {**

**if (arr[i][j] == 1) g++;**

**}**

**arr2[i] = new int[g];**

**arr2[i][0] = g;**

**for (int j = 0, x = 1; j != count; j++) {**

**if (arr[i][j] == 1) {**

**arr2[i][x] = j;**

**x++;**

**}**

**}**

**}**

**cout << endl << endl << endl;**

**cout << "Список смежности" << endl;**

**for (int i = 0; i < count; i++) {**

**for (int j = 1; j < arr2[i][0]; j++) {**

**cout << arr2[i][j] << ' ';**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < count; i++) {**

**for (int h = 0; h < count; h++) visited[h] = 0;**

**cout << "Вершина " << i << endl;**

**Queue.push(i);**

**while (!Queue.empty()) {**

**int node = Queue.front();**

**Queue.pop();**

**if (visited[node] == 1) continue;**

**visited[node] = 1;**

**for (int j = 1; j < arr2[node][0]; j++) {**

**if (visited[arr2[node][j]] == 1) continue;**

**Queue.push(arr2[node][j]);**

**}**

**cout << node << ' ';**

**}**

**cout << endl;**

**}**

**/\***

**\* 2.1 Для матричной формы представления графов реализуйте**

**алгоритм обхода в ширину с использованием очереди, построенной**

**на основе структуры данных «список», самостоятельно созданной**

**в лабораторной работе № 3.**

**\*/**

**cout << endl << endl << endl;**

**cout << "Список смежности для спец.листа" << endl << endl;**

**queuee<int> b;**

**for (int i = 0; i < count; i++) {**

**start = clock();**

**for (int h = 0; h < count; h++) visited[h] = 0;**

**cout << "Вершина " << i << endl;**

**List<int>\* g = new List<int>(i);**

**b.Push(g);**

**while (b.at(0) != -1) {**

**int node = b[0];**

**b.pop();**

**if (visited[node] == 1) continue;**

**visited[node] = 1;**

**for (int j = 0; j < count; j++) {**

**if (arr[node][j] == 0 || visited[j] == 1) continue;**

**List<int>\* tmp = new List<int>(j);**

**b.Push(tmp);**

**}**

**cout << node << ' ';**

**}**

**double time = 0.0;**

**end = clock();**

**time += (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("\nВремя работы: %lf в секундах\n\n", time);**

**}**

**}**

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

Рис. 1

Результат получился правильный, все действия совершены верно.

Результат работы программы, показанный на рисунке 1.

**Задание 2:**

**Листинг**

**//L\_ptr.cpp**

**#include "L\_ptr.h"**

**template<typename T>**

**void L\_ptr<T>::Display() {**

**List<T>\* x = head;**

**while (x != nullptr) {**

**std::cout << x->number << std::endl;**

**x = x->next;**

**}**

**}**

**template<typename T>**

**int L\_ptr<T>::at(int count) {**

**if (this->size == 0 || count < 0 || count > size) return -1;**

**else {**

**return this->operator [](count);**

**}**

**}**

**template<typename T>**

**List<T>\* L\_ptr<T>::\_\_index\_\_(int count) {**

**if (count < 0 || count >= size) nullptr;**

**int id = 0;**

**List<T>\* tmp = head;**

**while (id + 1 != count) {**

**tmp = tmp->next;**

**id++;**

**}**

**return tmp;**

**}**

**template<typename T>**

**void L\_ptr<T>::Del\_vertex(int count) {**

**size--;**

**if (count == 0) {**

**List<T>\* tmp = head;**

**head = head->next;**

**tmp = nullptr;**

**}**

**else {**

**List<T>\* tmp = \_\_index\_\_(count);**

**tmp->next = tmp->next->next;**

**}**

**};**

**template<typename T>**

**int L\_ptr<T>::lenght() { return size; };**

**template<typename T>**

**List<T>\* top() {**

**if (this->size <= 0) {**

**std::cout << "Данных нет" << std::endl;**

**\_sleep(3000);**

**abort();**

**}**

**return this->head;**

**};**

**template<typename T>**

**void L\_ptr<T>::pop() {**

**if (this->size <= 0) {**

**std::cout << "Данных нет" << std::endl;**

**}**

**List<T>\* tmp = this->head;**

**this->head = this->head->next;**

**tmp->next = nullptr;**

**this->size--;**

**delete tmp;**

**};**

**template<typename T>**

**List<T>\* L\_ptr<T>::find(T data) {**

**if (size == 0) {**

**abort();**

**}**

**List<T>\* tmp = this->head;**

**while (tmp->number != data) {**

**if (tmp->next == nullptr) {**

**return this->head;**

**}**

**tmp = tmp->next;**

**}**

**return tmp;**

**}**

**template<typename T>**

**int L\_ptr<T>::operator [] (int a) {**

**List<T>\* x = head;**

**if (size == 0) {**

**std::cout << "Лист пуст" << std::endl;**

**return 0;**

**}**

**while (a != 0) {**

**x = x->next;**

**a--;**

**if (a != 0 && x == nullptr || a != 0 && x->next == nullptr || a == 0 && x == nullptr) {**

**std::cout << "Ошибка скобок" << std::endl;**

**return 0;**

**}**

**}**

**return x->number;**

**}**

**template<typename T>**

**void L\_ptr<T>::Swap(int top1, int top2) {**

**List<T>\* head\_tmp1 = nullptr;**

**List<T>\* head\_tmp2 = nullptr;**

**List<T>\* tmp1 = nullptr, \* tmp2 = nullptr;**

**List<T>\* x = head;**

**if (top1 == 0) {**

**tmp1 = head;**

**head\_tmp1 = head;**

**}**

**if (top2 >= size) {**

**std::cout << "Ошибка скобок" << std::endl;**

**return;**

**}**

**while (top2 != 0) {**

**if (top1 == 1) { head\_tmp1 = x; tmp1 = x->next; }**

**if (top2 == 1) { head\_tmp2 = x; tmp2 = x->next; }**

**x = x->next;**

**top1--; top2--;**

**}**

**int g;**

**tmp2 = tmp1;**

**tmp1 = x;**

**//**

**g = tmp2->number;**

**tmp2->number = tmp1->number;**

**tmp1->number = g;**

**}**

**//**

**//**

**// L\_ptr.h**

**#pragma once**

**// L\_ptr.h**

**#include "List.cpp"**

**#include <iostream>**

**template <typename T>**

**class L\_ptr {**

**protected:**

**int size;**

**List<T>\* head = nullptr;**

**List<T>\* end = nullptr;**

**List<T>\* \_\_index\_\_(int count);**

**public:**

**L\_ptr() : size(0) { };**

**L\_ptr(int size) : size(size) { };**

**int operator [] (int a);**

**const List<T>\* Get\_head() { return head; };**

**List<T>\* top();**

**void pop();**

**// находим элемент по данным**

**List<T>\* find(T data);**

**// удаление узла**

**void Del\_vertex(int count);**

**// меняем местами два узла**

**void Swap(int top1, int top2);**

**// проход по списку**

**int at(int count);**

**// получить длину списка**

**int lenght();**

**// вывести все на экран**

**void Display();**

**};**

**//**

**//**

**// List.cpp**

**template <typename T>**

**class L\_ptr;**

**template <typename T>**

**class List {**

**public:**

**List\* next = nullptr;**

**T number;**

**// конструктор**

**List() : number(0), next(nullptr) { };**

**List(int number) : number(number), next(nullptr) { };**

**List(const List& a) {**

**this->number = a.number;**

**this->next = a.next;**

**};**

**// булевые операторы**

**bool operator == (List& a) { return number == a.number; };**

**bool operator >= (List& a) { return number >= a.number; };**

**bool operator <= (List& a) { return number <= a.number; };**

**bool operator < (List& a) { return number < a.number; };**

**bool operator > (List& a) { return number > a.number; };**

**bool operator == (int a) { return number == a; };**

**bool operator >= (int a) { return number >= a; };**

**bool operator <= (int a) { return number <= a; };**

**bool operator < (int a) { return number < a; };**

**bool operator > (int a) { return number > a; };**

**//**

**T& operator = (List\* a) { //**

**this->number = a->number;**

**return this->number;**

**};**

**operator int() { return this->number; };**

**~List() { delete next; next = nullptr; }**

**friend L\_ptr<T>;**

**};**

**//**

**//**

**// laba\_8.cpp**

**#include <iostream>**

**#include <locale>**

**#include <queue>**

**#include "L\_ptr.cpp"**

**using namespace std;**

**template <typename T>**

**class queuee : public L\_ptr<T> {**

**public:**

**queuee() : L\_ptr<T>() { };**

**void Push(List<T>\* g) {**

**List<T>\* tmp = new List<T>;**

**\*tmp = g;**

**if (this->size == 0) {**

**this->end = tmp;**

**this->head = tmp;**

**}**

**else {**

**this->end->next = tmp;**

**this->end = tmp;**

**}**

**this->size++;**

**};**

**};**

**int main()**

**{**

**setlocale(LC\_ALL, "RUS");**

**srand(time(NULL));**

**/\***

**1. Сгенерируйте (используя генератор случайных чисел) матрицу**

**смежности для неориентированного графа G. Выведите матрицу на экран.**

**\*/**

**int count;**

**cout << "Введите размер матрицы" << endl;**

**cin >> count;**

**cout << endl << endl;**

**cout << "Матрица смежности" << endl;**

**int\*\* arr = new int\* [count];**

**for (int i = 0; i != count; i++) {**

**arr[i] = new int[count];**

**for (int j = 0; j != count; j++) {**

**arr[i][j] = 0;**

**}**

**}**

**for (int i = 0; i < count; i++) {**

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

**int r = rand() % 101;**

**if (r > 40) {**

**arr[i][j] = 1;**

**arr[j][i] = 1;**

**}**

**else {**

**arr[i][j] = 0;**

**arr[j][i] = 0;**

**}**

**arr[i][i] = 0;**

**}**

**}**

**for (int i = 0; i != count; i++) {**

**for (int j = 0; j != count; j++) {**

**cout << arr[i][j] << ' ';**

**}**

**cout << endl;**

**}**

**cout << endl << endl;**

**/\***

**2. Для сгенерированного графа осуществите**

**процедуру обхода в ширину**

**\*/**

**clock\_t start, end; // объявляем переменные для определения времени выполнения**

**queue<int> Queue;**

**int\* visited = new int[count];**

**for (int i = 0; i != count; i++) {**

**start = clock();**

**for (int i = 0; i != count; i++) visited[i] = 0;**

**Queue.push(i);**

**cout << "Вершина " << i << endl;**

**while (!Queue.empty()) {**

**int node = Queue.front();**

**Queue.pop();**

**if (visited[node] == 1) continue;**

**visited[node] = 1;**

**for (int i = 0; i != count; i++) {**

**if (arr[node][i] == 0 || visited[i] == 1) continue;**

**Queue.push(i);**

**}**

**cout << node << ' ';**

**}**

**cout << endl;**

**double time = 0.0;**

**end = clock();**

**time += (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("Время работы: %lf в секундах\n\n", time);**

**}**

**/\***

**3. Реализуйте процедуру обхода в ширину для**

**графа, представленного списками смежности.**

**\*/**

**int\*\* arr2 = new int\*[count];**

**for (int i = 0; i < count; i++) {**

**int g = 1;**

**for (int j = 0; j < count; j++) {**

**if (arr[i][j] == 1) g++;**

**}**

**arr2[i] = new int[g];**

**arr2[i][0] = g;**

**for (int j = 0, x = 1; j != count; j++) {**

**if (arr[i][j] == 1) {**

**arr2[i][x] = j;**

**x++;**

**}**

**}**

**}**

**cout << endl << endl << endl;**

**cout << "Список смежности" << endl;**

**for (int i = 0; i < count; i++) {**

**for (int j = 1; j < arr2[i][0]; j++) {**

**cout << arr2[i][j] << ' ';**

**}**

**cout << endl;**

**}**

**for (int i = 0; i < count; i++) {**

**for (int h = 0; h < count; h++) visited[h] = 0;**

**cout << "Вершина " << i << endl;**

**Queue.push(i);**

**while (!Queue.empty()) {**

**int node = Queue.front();**

**Queue.pop();**

**if (visited[node] == 1) continue;**

**visited[node] = 1;**

**for (int j = 1; j < arr2[node][0]; j++) {**

**if (visited[arr2[node][j]] == 1) continue;**

**Queue.push(arr2[node][j]);**

**}**

**cout << node << ' ';**

**}**

**cout << endl;**

**}**

**/\***

**\* 2.1 Для матричной формы представления графов реализуйте**

**алгоритм обхода в ширину с использованием очереди, построенной**

**на основе структуры данных «список», самостоятельно созданной**

**в лабораторной работе № 3.**

**\*/**

**cout << endl << endl << endl;**

**cout << "Список смежности для спец.листа" << endl << endl;**

**queuee<int> b;**

**for (int i = 0; i < count; i++) {**

**start = clock();**

**for (int h = 0; h < count; h++) visited[h] = 0;**

**cout << "Вершина " << i << endl;**

**List<int>\* g = new List<int>(i);**

**b.Push(g);**

**while (b.at(0) != -1) {**

**int node = b[0];**

**b.pop();**

**if (visited[node] == 1) continue;**

**visited[node] = 1;**

**for (int j = 0; j < count; j++) {**

**if (arr[node][j] == 0 || visited[j] == 1) continue;**

**List<int>\* tmp = new List<int>(j);**

**b.Push(tmp);**

**}**

**cout << node << ' ';**

**}**

**double time = 0.0;**

**end = clock();**

**time += (double)(end - start) / CLOCKS\_PER\_SEC;**

**printf("\nВремя работы: %lf в секундах\n\n", time);**

**}**

**}**

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

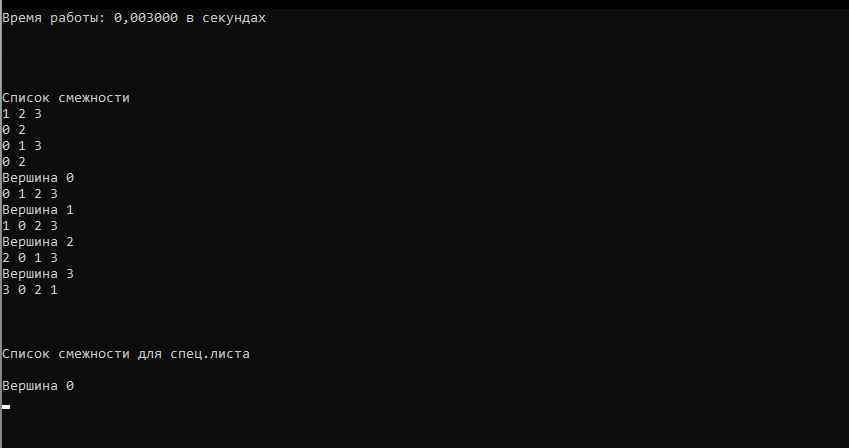
****

Рис. 1

Результат получился правильный, все действия совершены верно.

Результат работы программы, показанный на рисунке 1.