Министерство образования и науки Российской Федерации

Пензенский государственный университет

Факультет вычислительной техники

**Отчёт**

По лабораторной работе №5

По курсу "ЛиОАвИЗ"

Выполнили:

Студент группы 20ВВ1

Репин И.В.

Приняла:

Юрова О.В.

Пенза 2021

**Название:** Обход графа в ширину

**Цель работы:** выполнить ряд заданий

**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

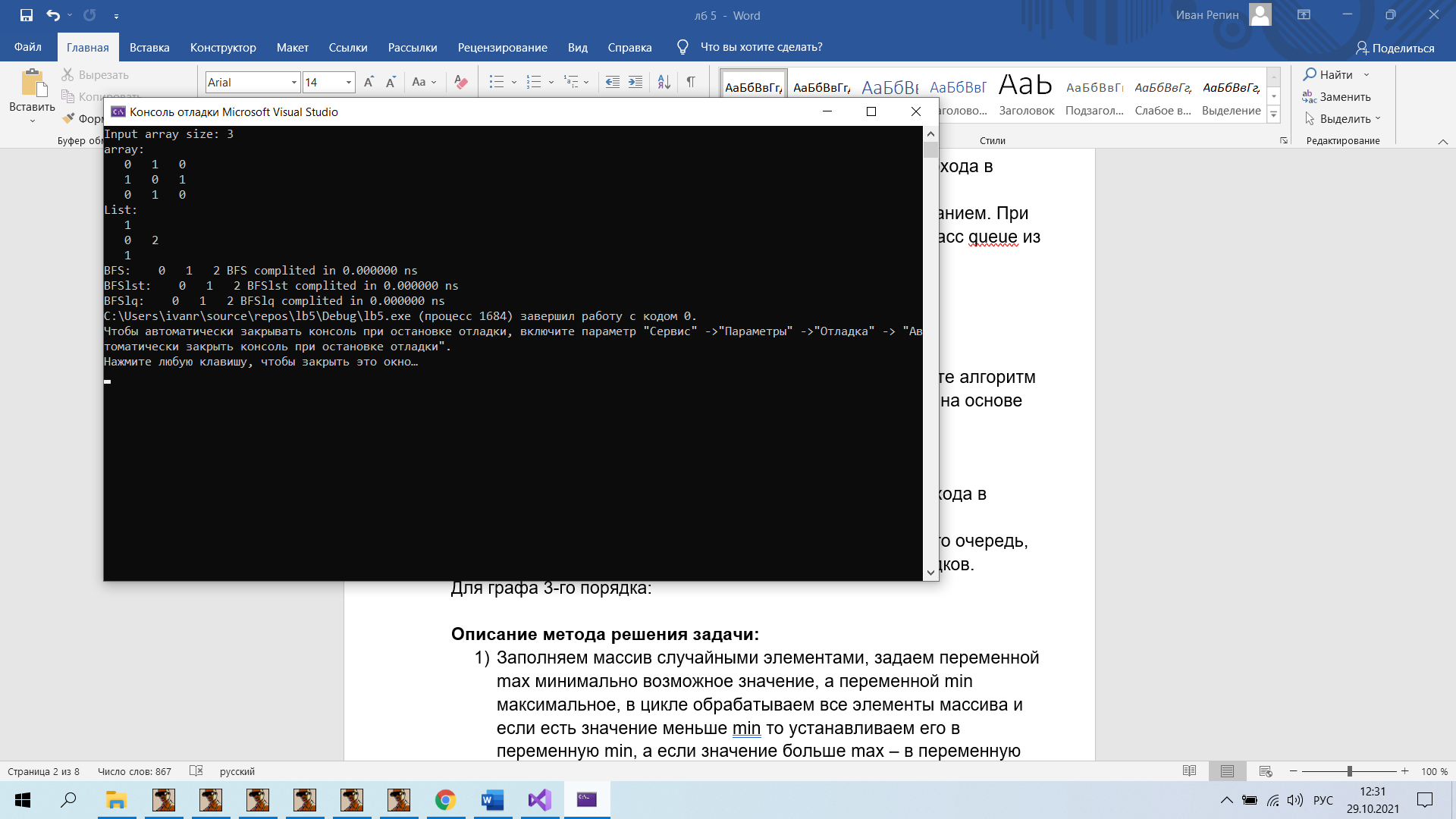
работе № 3.

2. Оцените время работы двух реализаций алгоритмов обхода в ширину

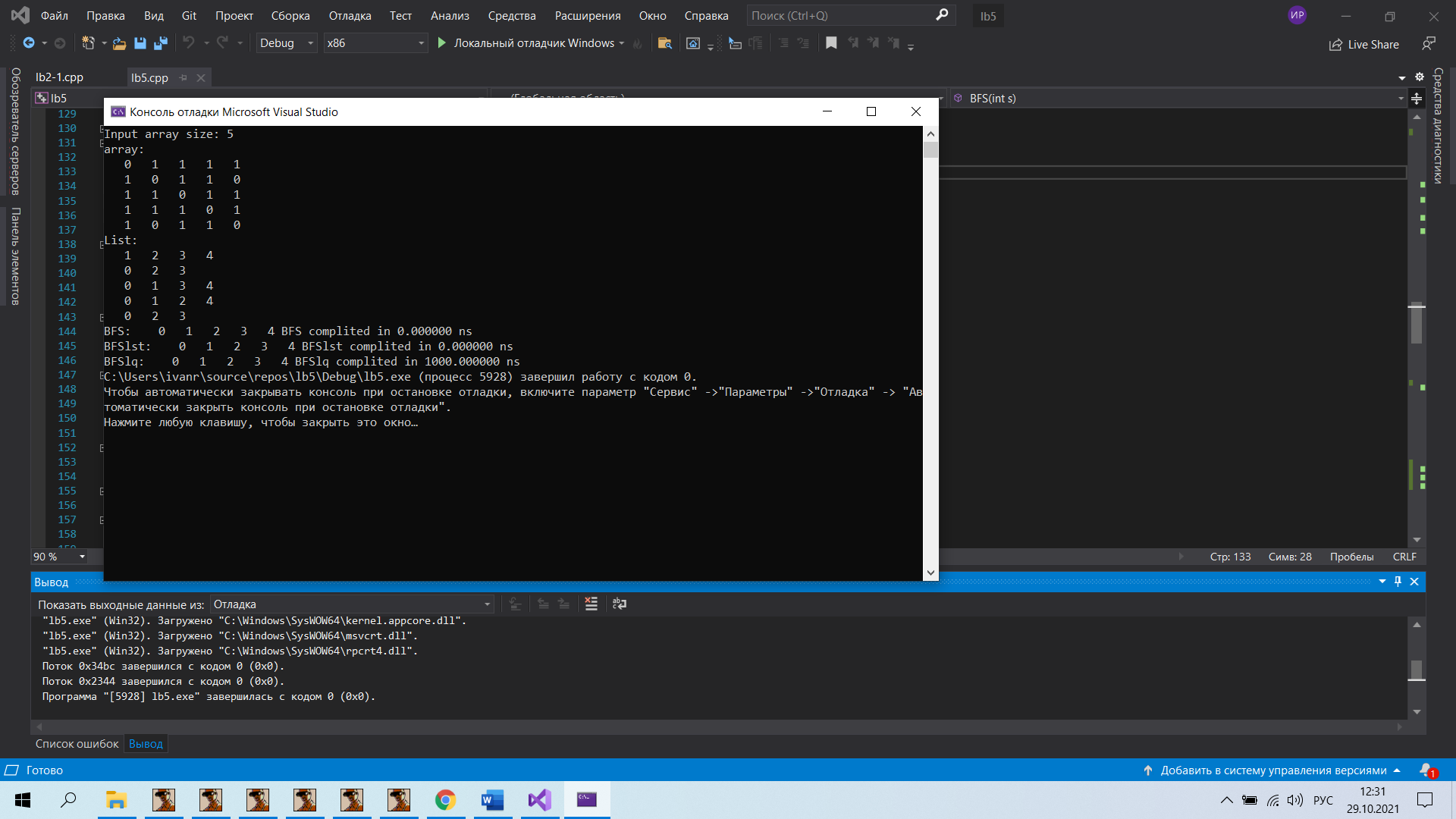
(использующего стандартный класс queue и использующего очередь,

реализованную самостоятельно) для графов разных порядков.

Для графа 3-го порядка:



Для графа 5-го порядка:



**Описание метода решения задачи:**

**Задание 1**

1. Заполняем массив случайными элементами, задаем переменной max минимально возможное значение, а переменной min максимальное, в цикле обрабатываем все элементы массива и если есть значение меньше min то устанавливаем его в переменную min, а если значение больше max – в переменную max
2. Реализация обхода в ширину для матрицы:

Вход: G – матрица смежности графа.

Выход: номера вершин в порядке их прохождения на экране.

Алгоритм ПОШ

1.1. для всех i положим NUM[i] = False пометим как &quot;не посещенную&quot;;

1.2. ПОКА существует &quot;новая&quot; вершина v

1.3. ВЫПОЛНЯТЬ BFS (v).

Алгоритм BFS(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. пометить v как &quot;посещенную&quot; NUM[v] = True;

2.4. ПОКА Q != Æ очередь не пуста ВЫПОЛНЯТЬ

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

2.9. ЕСЛИ G(v,i) = = 1И NUM[i] = = False

2.10. ТО

2.11. Поместить i в очередь Q.push(i);

2.12. пометить v как &quot;посещенную&quot; NUM[v] = True;

1. Аналогично для списка

**Задание 2**

1. Новые элементы очереди добавляем в конец списка, берем первый элемент из начала списка, удаляем первый элемент из списка путем перестановки указателей

**Листинг:**

// lb5.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <queue>

#include <time.h>;

using namespace std;

int\*\* m;

int num;

int\* vis;

int\* vislst;

int\* vislq;

queue <int> q;

void BFS(int s);

struct list\* init(int a);

struct list\* addelem(list\* lst, int number, int id, int line);

struct lq\* initlq(int a);

struct lq\* addelemlq(lq\* lst, int field);

void listprint(list\* lst);

struct lq\* poplq(lq\* lst);

void BFSlst(int s, list\* lst);

void BFSlq(int s);

struct list

{

int field, id, line; // поле данных

struct list\* ptr; // указатель на следующий элемент

};

struct lq

{

int field; // поле данных

struct lq\* ptr; // указатель на следующий элемент

};

struct list\* init(int a) // а- значение первого узла

{

struct list\* lst;

// выделение памяти под корень списка

lst = (struct list\*)malloc(sizeof(struct list));

lst->field = a;

lst->id = a;

lst->line = -1;

lst->ptr = NULL; // это последний узел списка

return(lst);

}

struct lq\* initlq(int a) // а- значение первого узла

{

struct lq\* lst;

// выделение памяти под корень списка

lst = (struct lq\*)malloc(sizeof(struct lq));

lst->field = a;

lst->ptr = NULL; // это последний узел списка

return(lst);

}

struct lq\* addelemlq(lq\* lst, int field) {

struct lq\* temp;

struct lq\* p;

temp = (struct lq\*)malloc(sizeof(lq));

while (lst->ptr != NULL) // просматриваем список начиная с корня

{ // пока не найдем узел, предшествующий NULL

lst = lst->ptr;

}

lst->ptr = temp; // предыдущий узел указывает на создаваемый

temp->field = field; // сохранение поля данных добавляемого узла

temp->ptr = NULL; // созданный узел указывает на следующий элемент

return(temp);

}

struct list\* addelem(list\* lst, int number, int id, int line)

{

struct list\* temp, \* p;

temp = (struct list\*)malloc(sizeof(list));

p = lst->ptr; // сохранение указателя на следующий узел

lst->ptr = temp; // предыдущий узел указывает на создаваемый

temp->field = number; // сохранение поля данных добавляемого узла

temp->id = id;

temp->line = line;

temp->ptr = p; // созданный узел указывает на следующий элемент

return(temp);

}

void listprint(list\* lst)

{

struct list\* p;

p = lst->ptr;

int line, id;

line = 0;

id = 0;

do {

if (p->line > line) {

printf("\n");

line++;

id = 0;

}

printf("%4i", p->id);

p = p->ptr; // переход к следующему узлу

} while (p != NULL);

}

struct lq\* poplq(lq\* lst)

{

struct lq\* temp;

temp = lst;

while (temp->ptr->ptr != NULL) // просматриваем список начиная с корня

{ // пока не найдем узел, предшествующий NULL

temp = temp->ptr;

}

temp->ptr = NULL; // переставляем указатель

return(temp);

}

int frontlq(lq\* lst)

{

struct lq\* p;

p = lst;

while (p->ptr != NULL) {

p->field = p->ptr->field;

p = p->ptr;

}

printf("\nFront: %4i", p->field);

return(p->field);

}

void BFS(int s) {

vis[s] = 1;

q.push(s);

while (!q.empty()) {

int v = q.front();

q.pop();

printf("%4i", v);

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

if ((m[v][i] == 1) and (vis[i] == 0)) {

q.push(i);

vis[i] = 1;

}

}

}

}

void BFSlst(int s, list\* lst) {

q.push(s);

struct list\* p;

p = lst->ptr;

vislst[s] = 1;

while (p->line != s) {

p = p->ptr;

}

int i = 0;

while (!q.empty()) {

int v = q.front();

q.pop();

printf("%4i", v);

p = lst->ptr;

while (p->line != v) {

p = p->ptr;

}

do {

i = p->id;

if (vislst[i] == 0) {

q.push(i);

vislst[i] = 1;

}

p = p->ptr;

} while (p != NULL);

}

}

void BFSlq(int s) {

vislq[s] = 1;

struct lq\* lst = initlq(0);

struct lq\* p = lst;

addelemlq(lst, s);

while (lst->ptr != NULL) {

int v = lst->ptr->field;

//int v = frontlq(lst);

printf("%4i", v);

//poplq(lst);

lst->ptr = lst->ptr->ptr;

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

if ((m[v][i] == 1) and (vislq[i] == 0)) {

addelemlq(lst, i);

vislq[i] = 1;

}

}

}

}

int main()

{

//generate array

clock\_t start, end;

int generate;

printf("Input array size: ");

scanf("%i", &num);

m = new int\* [num];

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

m[i] = new int[num];

}

srand(time(NULL));

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

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

if (j == i) {

m[i][j] = 0;

}

else if (j > i) {

generate = rand() % 100;

if (generate >= 30) {

m[i][j] = 1;

}

else {

m[i][j] = 0;

}

}

else {

m[i][j] = m[j][i];

}

}

}

//print array

printf("array:");

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

printf("\n");

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

printf("%4i", m[i][j]);

}

}

//generate list

struct list\* lst = init(0);

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

for (int j = (num - 1); j >= 0; j--) {

if (m[i][j] == 1) {

addelem(lst, 1, j, i);

}

}

}

//print list

printf("\nList: \n");

listprint(lst);

//rounds

vis = new int[num];

vislst = new int[num];

vislq = new int[num];

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

vis[i] = 0;

vislst[i] = 0;

vislq[i] = 0;

}

//result

start = clock();

printf("\nBFS: ");

BFS(0);

end = clock();

double time\_spent = (double)(end - start) \* 1000000 / CLOCKS\_PER\_SEC;

printf(" BFS complited in %f ns", time\_spent);

start = clock();

printf("\nBFSlst: ");

BFSlst(0, lst);

end = clock();

time\_spent = (double)(end - start) \* 1000000 / CLOCKS\_PER\_SEC;

printf(" BFSlst complited in %f ns", time\_spent);

start = clock();

printf("\nBFSlq: ");

BFSlq(0);

end = clock();

time\_spent = (double)(end - start) \* 1000000 / CLOCKS\_PER\_SEC;

printf(" BFSlq complited in %f ns", time\_spent);

}

**Результаты работы программы:**

Результаты работы программы представлен на рисунке 1

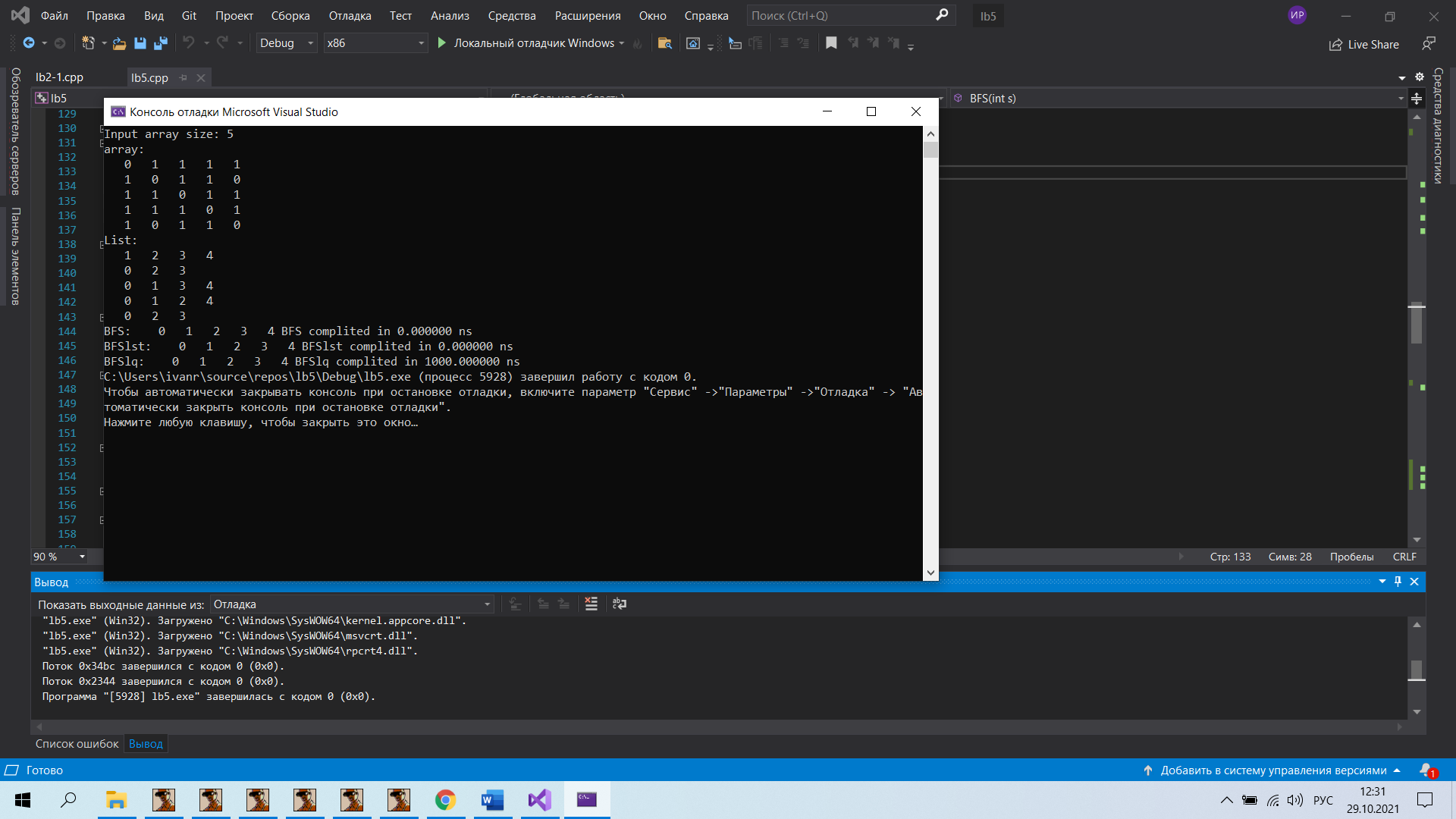


Рисунок 1

**Вывод:**

В ходе выполнения лабораторной работы освоили обход в ширину