Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в ширину»

Выполнила:

студентка группы 22ВВВ2

Расторгуева К.В.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Обход графа в ширину.

**Цель работы**

Изучение обхода графа в ширину, его реализация с помощью очереди.

**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.

**Пояснительный текст к программе**

Для начала работы программы пользователю необходимо ввести размер матрицы *n*. После чего происходит выделение памяти для матрицы смежности и её формирование с помощью генератора случайных чисел *rand()*. Затем происходит вывод матрицы. Функция *BFS()* просматривает все связи определенной вершины с другими вершинами и добавляет их в очередь Q. Затем для каждой вершины из очереди просматривается связь с другими вершинами. Учёт посещаемости вершин происходит в массиве *tops\_visited*[*n*]. Аналогичный алгоритм использован для обхода неориентированного графа, представленного списком смежности.

Назначение отдельных операторов программы указывается в виде комментариев. Программа завершает свою работу после нажатия на любую клавишу на клавиатуре.

**Результат выполнения программы**

Результат работы программы представлен на рис. 1-2.

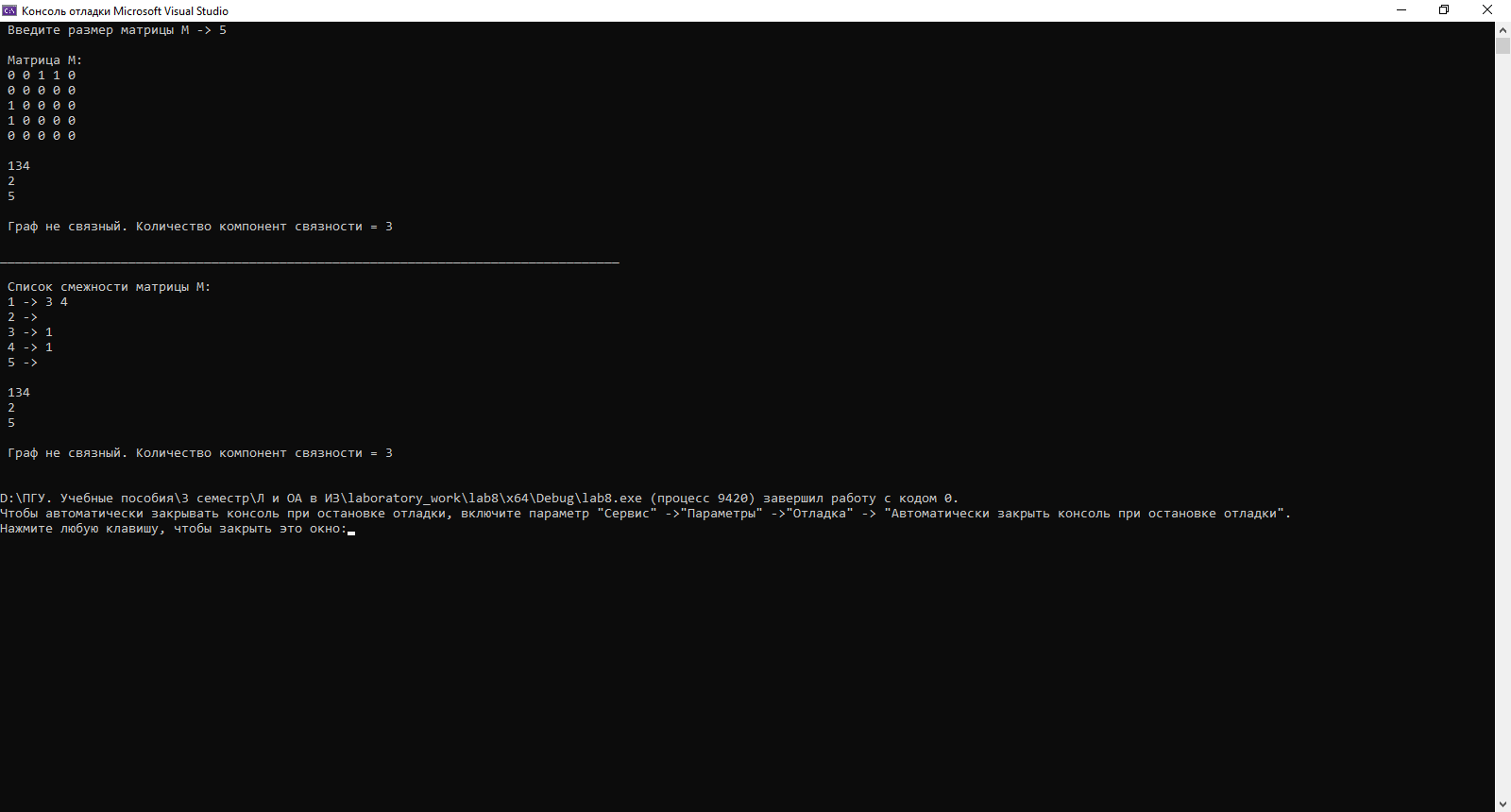
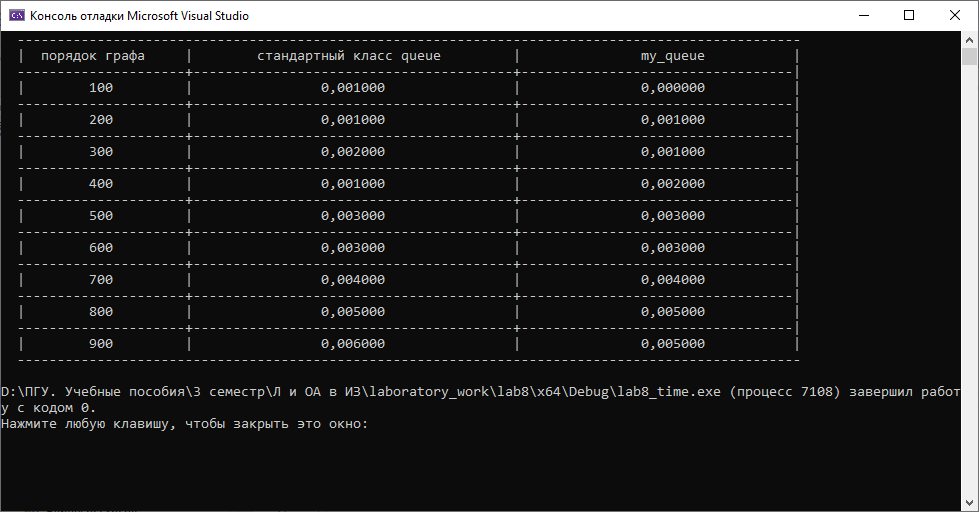


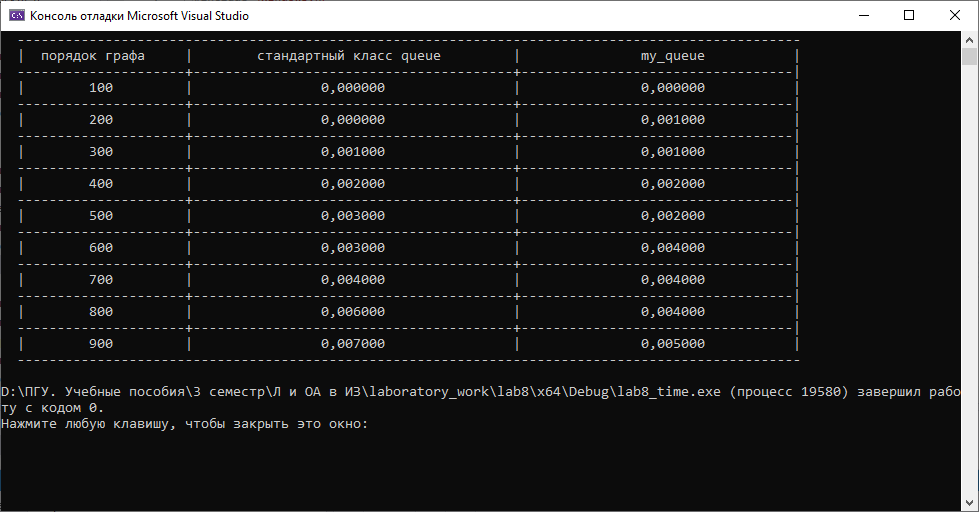
Рисунок 1 – алгоритм с использованием стандартного класса queue

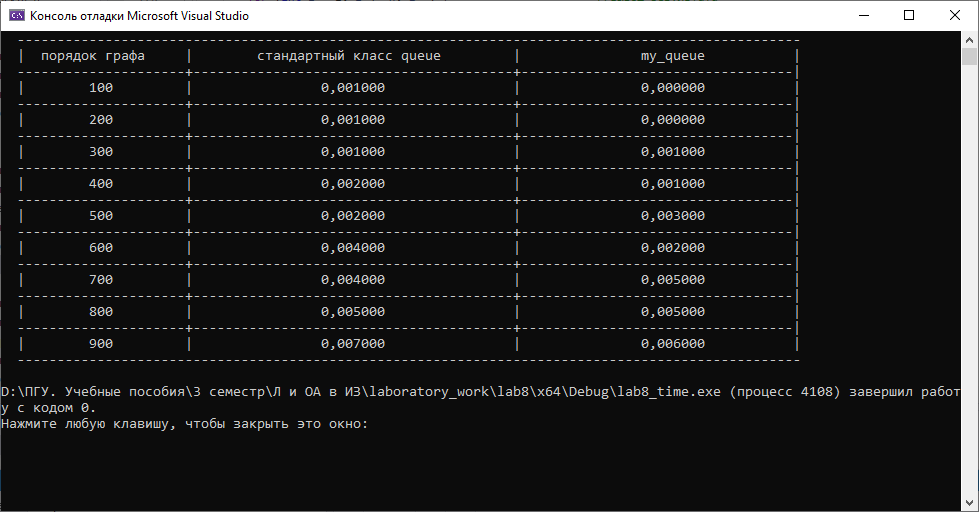


Рисунок 2 – алгоритм с использованием очереди, реализованной самостоятельно

**Оценка времени работы**







**Выводы**

В ходе выполнения лабораторной работы была разработана на языке Си программа, осуществляющая обход неориентированного графа в ширину, представленного как в матричном виде, так и в виде списков смежности. Была произведена оценка времени работы двух реализаций алгоритма в ширину (использующего стандартный класс queue и использующего очередь, реализованную самостоятельно) для графов разных порядков, которая выявила работу алгоритма с использованием очереди, реализованной самостоятельно, более эффективной. Так как для стандартный класс queue универсален и работает с различными данными. Для корректной работы сначала необходимо определение типов данных, с которыми производится работа, на что необходимо время.

**Приложение**

**Алгоритм с использованием стандартного класса queue**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <queue>

#include <iostream>

struct node {

int top;

struct node\* next, \* next\_list;

};

struct node\* head = NULL, \* head\_top = NULL;

void BFS(int i, int n, int\*\* G, int\* tops\_visited) {

int v;

int v\_start = i;

printf(" ");

using namespace std;

queue <int> Q;

Q.push(v\_start);

tops\_visited[v\_start] = 1;

while (!Q.empty()) {

v = Q.front();

Q.pop();

printf("%d", v);

for (int i = 1; i <= n; i++) {

if ((G[v][i] == 1) && (tops\_visited[i] == 0)) {

Q.push(i);

tops\_visited[i] = 1;

}

}

}

}

void BFS\_list(struct node\* head\_save, struct node\* head\_top, int\* tops\_visited\_4\_list) {

int v;

struct node\* tmp\_top = NULL;

struct node\* start\_head = head\_top;

printf(" ");

using namespace std;

queue <int> Q;

Q.push(start\_head->top);

tops\_visited\_4\_list[start\_head->top] = 1;

while (!Q.empty()) {

v = Q.front();

start\_head = head\_save;

while (start\_head != NULL){

if (start\_head->top == v) break;

start\_head = start\_head->next;

}

Q.pop();

printf("%d", start\_head->top);

tmp\_top = start\_head->next\_list;

while(tmp\_top != NULL){

if (tops\_visited\_4\_list[tmp\_top->top] == 0) {

Q.push(tmp\_top->top);

tops\_visited\_4\_list[tmp\_top->top] = 1;

}

tmp\_top = tmp\_top->next;

}

}

return;

}

struct node\* get\_struct(int\*\* G, int n) {

struct node\* p = NULL;

struct node\* tmp\_top = NULL;

struct node\* tmp = NULL;

struct node\* head\_top\_first = NULL;

int flag\_first = 0;

for (int i = 1; i <= n; i++) {

((p = (node\*)malloc(sizeof(struct node))) == NULL); // выделение памяти под новую вершину

p->top = i;

p->next = NULL;

if (flag\_first == 0) {

head\_top\_first = p;

head\_top = p;

flag\_first++;

tmp\_top = head\_top;

}

else {

tmp\_top->next = p;

tmp\_top = p;

}

}

tmp\_top->next = NULL;

for (int i = 1; i <= n; i++) {

flag\_first = 0;

for (int j = 1; j <= n; j++) {

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

((p = (node\*)malloc(sizeof(struct node))) == NULL); // выделение память под новый элемент вершины

p->top = j;

p->next = NULL;

if (flag\_first == 0) {

head = p;

flag\_first++;

tmp = head;

}

else {

tmp->next = p;

tmp = p;

}

}

}

head\_top->next\_list = head;

head\_top = head\_top->next;

head = NULL;

}

return head\_top\_first;

}

/\* Просмотр содержимого списка. \*/

void review(struct node\* head\_top) {

struct node\* tmp\_top = NULL;

if (head\_top == NULL) {

printf("Список пуст\n");

return;

}

while (head\_top != 0) {

printf(" %d -> ", head\_top->top);

tmp\_top = head\_top->next\_list;

while (tmp\_top != NULL) {

printf("%d ", tmp\_top->top);

tmp\_top = tmp\_top->next;

}

head\_top = head\_top->next;

printf("\n");

}

return;

}

int main() {

int n;

int comp\_sv = 0;

setlocale(0, "rus");

printf(" Введите размер матрицы М -> ");

scanf("%d", &n);

printf("\n");

int\*\* G = (int\*\*)malloc(n \* sizeof(int\*));

for (int i = 1; i <= n; i++) {

G[i] = (int\*)malloc(n \* sizeof(int));

}

int\* tops\_visited = (int\*)malloc(n \* sizeof(int\*));

for (int i = 1; i <= n; i++) {

tops\_visited[i] = 0;

}

int\* tops\_visited\_4\_list = (int\*)malloc(n \* sizeof(int\*));

for (int i = 1; i <= n; i++) {

tops\_visited\_4\_list[i] = 0;

}

printf(" Матрица М:\n");

srand(time(0));

for (int i = 1; i <= n; i++) { //формирование матрицы смежности М

for (int j = 1; j <= n; j++) {

if (j >= i) {

if (i == j) G[i][j] = 0;

else {

G[i][j] = rand() % 2;

G[j][i] = G[i][j];

}

}

printf(" %d", G[i][j]);

}

printf("\n");

}

for (int i = 1; i <= n; i++) { //вывод результата

if (tops\_visited[i] == 0) {

comp\_sv++;

printf("\n");

BFS(i, n, G, tops\_visited);

}

}

if (comp\_sv == 1) printf("\n\n Граф связный\n\n");

else printf("\n\n Граф не связный. Количество компонент связности = %d", comp\_sv);

printf("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

printf(" Список смежности матрицы М:\n");

head = get\_struct(G, n);

review(head); //формирование списка смежности матрицы М

comp\_sv = 0;

head\_top = head;

while (head\_top != NULL) { //вывод результата

if (tops\_visited\_4\_list[head\_top->top] == 0) {

comp\_sv++;

printf("\n");

BFS\_list(head, head\_top, tops\_visited\_4\_list);

}

head\_top = head\_top->next;

}

if (comp\_sv == 1) printf("\n\n Граф связный\n\n");

else printf("\n\n Граф не связный. Количество компонент связности = %d\n\n", comp\_sv);

return 0;

}

**Алгоритм с использованием очереди, реализованной самостоятельно**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <iostream>

struct node {

int inf;

struct node\* next, \* prev;

};

typedef node\* pnode;

struct queue {

pnode head, tail;

};

/\* Последовательное добавление в список элемента\*/

queue\* push(queue& S, int v) {

pnode newnode;

newnode = new node;

newnode->inf = v;

if (S.head == NULL) {

newnode->next = NULL;

newnode->prev = NULL;

S.tail = newnode;

S.head = newnode;

}

else {

S.tail->next = newnode;

newnode->prev = S.tail;

S.tail = newnode;

S.tail->next = NULL;

}

return (&S);

}

/\* Удаление элемента по содержимому. \*/

queue\* pop(queue& S) {

if (S.head == NULL) {

printf("Список пуст\n");

}

pnode HeadNode = S.head;

S.head = HeadNode->next;

if (S.head) {

S.head->prev = NULL;

}

else S.head = NULL;

HeadNode->next = NULL;

delete HeadNode;

//printf("Операция выполнена успешно\n");

return(&S);

}

void BFS(int i, int n, int\*\* G, int\* tops\_visited) {

queue\* Q = (queue\*)malloc(sizeof(queue));

int v\_start = i;

Q->head = NULL;

Q->tail = NULL;

int v;

printf(" ");

Q = push(\*Q, v\_start);

tops\_visited[v\_start] = 1;

while (Q->head != NULL) {

v = Q->head->inf;

Q = pop(\*Q);

printf("%d", v);

for (int i = 1; i <= n; i++) {

if ((G[v][i] == 1) && (tops\_visited[i] == 0)) {

Q = push(\*Q, i);

tops\_visited[i] = 1;

}

}

}

}

int main() {

int n;

int comp\_sv = 0;

setlocale(0, "rus");

printf(" Введите размер матрицы М -> ");

scanf("%d", &n);

printf("\n");

int\*\* G = (int\*\*)malloc(n \* sizeof(int\*));

for (int i = 1; i <= n; i++) {

G[i] = (int\*)malloc(n \* sizeof(int));

}

int\* tops\_visited = (int\*)malloc(n \* sizeof(int\*));

for (int i = 1; i <= n; i++) {

tops\_visited[i] = 0;

}

int\* tops\_visited\_4\_list = (int\*)malloc(n \* sizeof(int\*));

for (int i = 1; i <= n; i++) {

tops\_visited\_4\_list[i] = 0;

}

printf(" Матрица М:\n");

srand(time(0));

for (int i = 1; i <= n; i++) { //формирование матрицы смежности М

for (int j = 1; j <= n; j++) {

if (j >= i) {

if (i == j) G[i][j] = 0;

else {

G[i][j] = rand() % 2;

G[j][i] = G[i][j];

}

}

printf(" %d", G[i][j]);

}

printf("\n");

}

for (int i = 1; i <= n; i++) { //вывод результата

if (tops\_visited[i] == 0) {

comp\_sv++;

printf("\n");

BFS(i, n, G, tops\_visited);

}

}

if (comp\_sv == 1) printf("\n\n Граф связный\n\n");

else printf("\n\n Граф не связный. Количество компонент связности = %d", comp\_sv);

return 0;

}