МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №8  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Обход графа в ширину»

Выполнили:

Студенты группы 23ВВВ2

Лисов Е.А.

Кочегин В.В.

Приняли:

Митрохин М. А.  
Юрова О.В.

Пенза 2024

**Цель работы**

Изучение обхода графа в ширину. Закрепление навыков его генерации, реализации обхода.

**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.

**Ход работы**

**Задание 1.**

**1** Реализовали код, генерирующий граф:

void createG(int\*\* G, int N) {

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

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

int edge = rand() % 2;

G[i][j] = edge;

G[j][i] = edge;

}

}

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

G[i][i] = 0;

}

}

**2** Для сгенерированного графа осуществили процедуру обхода в ширину, реализованную в соответствии с приведенным выше описанием. При реализации алгоритма в качестве очереди использовали класс **queue** из стандартной библиотеки С++:

void DFS(int\*\* G, int N, int\* NUM, int v) {

int\* stack = (int\*)malloc(N \* sizeof(int));

int top = -1;

stack[++top] = v;

while (top != -1) {

int vec = stack[top--];

if (NUM[vec] == 0) {

NUM[vec] = 1;

printf("%d ", vec + 1);

}

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

if (G[vec][i] == 1 && NUM[i] == 0) {

stack[++top] = i;

}

}

}

}

**3\*** Реализовали процедуру обхода в ширину для графа, представленного списками смежности:

void BFS(struct Graph\* graph, int start) {

bool visited[MAX\_VERTICES] = { false };

int queue[MAX\_VERTICES];

int front = 0, rear = 0;

visited[start] = true;

queue[rear++] = start;

printf("Результат обхода в ширину: : ", start + 1);

while (front < rear) {

int currentVertex = queue[front++];

printf("%d ", currentVertex + 1);

struct node\* temp = graph->adjLists[currentVertex];

while (temp) {

int adjVertex = temp->vertex;

if (!visited[adjVertex]) {

visited[adjVertex] = true;

queue[rear++] = adjVertex;

}

temp = temp->next;

}

}

printf("\n");

}

**Задание 2.**

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

void BFSqueue(vector<vector<int>> G, int start) {

int n = G.size();

vector<bool> visited(n, false);

queue<int> Q;

Q.push(start);

visited[start] = true;

while (!Q.empty()) {

int v = Q.front();

Q.pop();

cout << v + 1 << " ";

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

if (G[v][i] == 1 && !visited[i]) {

Q.push(i);

visited[i] = true;

}

}

}

}

void enqueue(int value) {

struct node\* p = (struct node\*)malloc(sizeof(struct node));

if (p == NULL) {

printf("Îøèáêà ïðè ðàñïðåäåëåíèè ïàìÿòè\n");

exit(1);

}

p->inf = value;

p->next = NULL;

if (last == NULL) {

head = p;

last = p;

}

else {

last->next = p;

last = p;

}

}

int isQueueEmpty(void) {

return head == NULL;

}

int dequeue(void) {

if (isQueueEmpty()) {

printf("Î÷åðåäü ïóñòà\n");

return -1;

}

struct node\* temp = head;

int value = temp->inf;

head = head->next;

if (head == NULL) {

last = NULL;

}

free(temp);

return value;

}

void BFS(int\*\* G, int n, int start) {

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

for (int i = 0; i < n; i++) visited[i] = 0;

enqueue(start);

visited[start] = 1;

while (!isQueueEmpty()) {

int v = dequeue();

printf("%d ", v + 1);

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

if (G[v][i] == 1 && !visited[i]) {

enqueue(i);

visited[i] = 1;

}

}

}

free(visited);

}

### Результаты работы программы

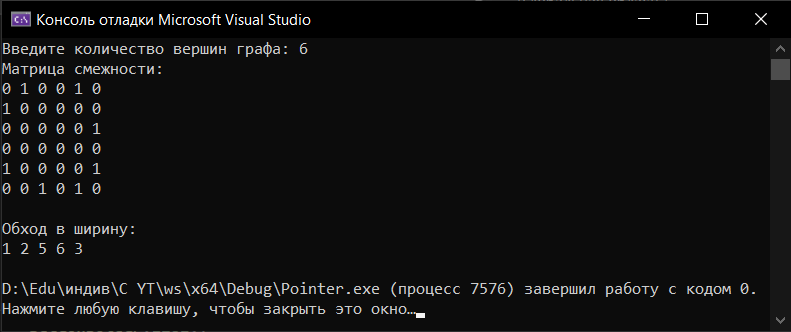


Рисунок 1 — Результаты работы программы для заданий 1 без \*

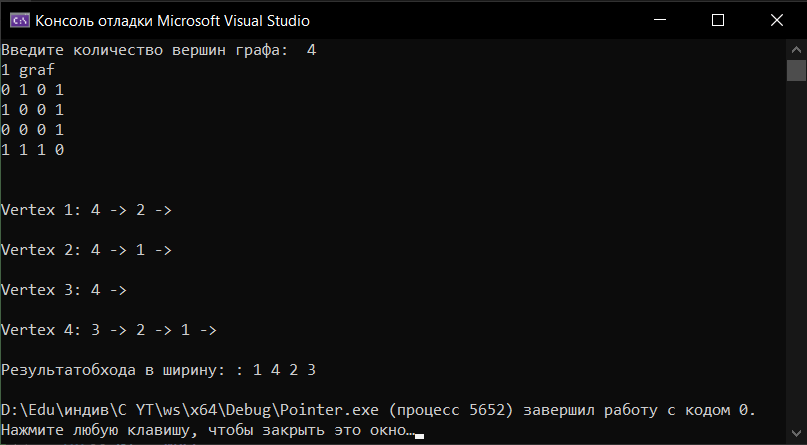
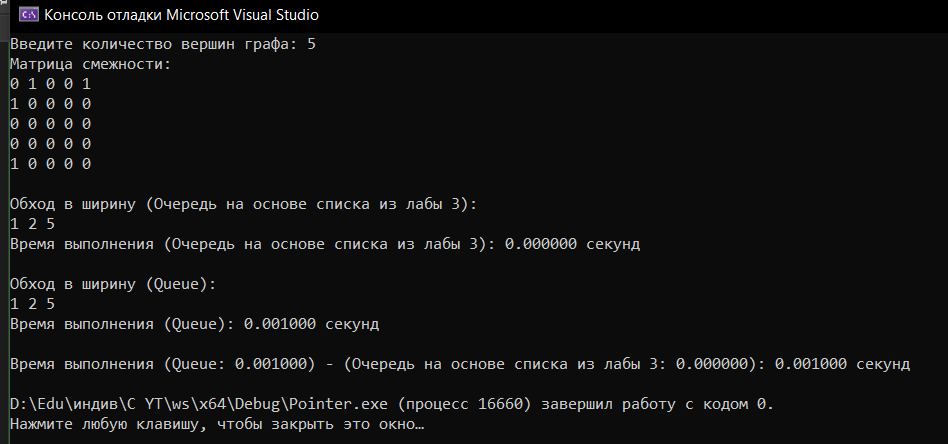


Рисунок 2 — Результаты работы программы для заданий 1 со \*

  
Рисунок 3 — Результаты работы программы для задания 2

### Вывод

В ходе выполнения работы были изучен принцип обхода графа в ширину и его реализация (не рекурсивная и рекурсивная).