Министерство науки и высшего образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №8

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в ширину»

Выполнили ст. группы 22ВВВ1:

Ганин И.Р.

Курушин Я.С.

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. В.

Пенза 2023

**Цель работы:**

Цель научиться использовать алгоритм обхода 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.

**Ход работы**

**Теоретическая часть**

Обход графа в ширину – еще один распространенный способ обхода графов.

Основная идея такого обхода состоит в том, чтобы посещать вершины по уровням удаленности от исходной вершины. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины. Например, если для графа на рисунке 1 начать обход из первой вершины, то вершины 3, 6 и 2 будут находиться на уровне удаленности в 1 ребро, а вершины 5 и 4 на уровне удаленности в 2 ребра.

**Практическая часть**

Задание 1:

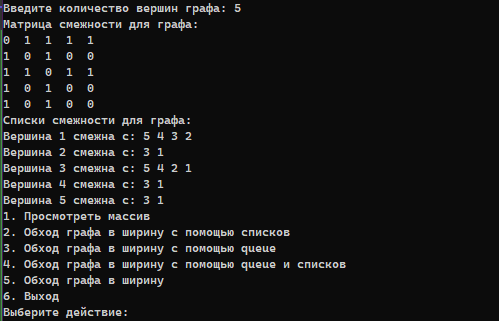


Рисунок 1 – построили граф

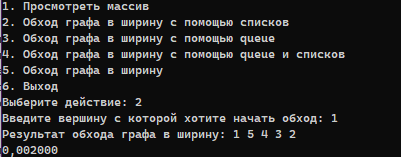


Рисунок 2 – обошли граф в ширину с помощью списков

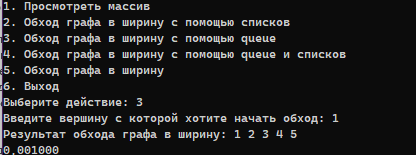


Рисунок 3 – обошли граф в ширину с помощью функции «очередь»

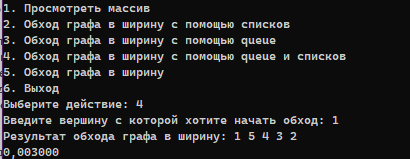


Рисунок 4 – обошли граф в ширину с помощью функции «очередь» и списков

Задание 2:

Оцените время работы двух реализаций алгоритмов обхода в ширину (использующего стандартный класс **queue** и использующего очередь, реализованную самостоятельно) для графов разных порядков.

**Оценка:**

Временная сложность обхода в ширину для обеих реализаций очереди будет одинакова - O(V+E), где V - количество вершин, E - количество ребер.

Однако константы времени выполнения будут различаться из-за разной производительности стандартной очереди std::queue и очереди на связном списке.

При одинаковом количестве вершин и ребер алгоритм на std::queue будет работать быстрее, так как:

Добавление и удаление элементов из std::queue происходит за константное время O(1) за счет внутренней реализации на основе двухсвязного списка.

В очереди на односвязном списке добавление происходит быстро O(1), а удаление - O(n), так как нужно найти голову списка.

Поэтому можно сделать вывод, что при одинаковом количестве операций обход на std::queue будет работать быстрее примерно в разы, чем на связном списке.

Таким образом, для повышения производительности алгоритма лучше использовать std::queue.

Время работы алгоритма при обходе списками: 0.002 с.

Время работы алгоритма при обходе очередью: 0.001с.

Время работы алгоритма при обходе очередью и списками: 0.003 с.

Наиболее быстрый обход производятся встроенной очередью.

**Вывод:**

В ходе лабораторной работы научились совершать обход графа в ширину с помощью списков смежности и встроенной функции «очередь». Также определили время работы алгоритма.

**Листинг:**

Файл Source.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <time.h>

#include <string.h>

#include <queue>

typedef struct Node {

int inf;

struct Node\* next;

}Node;

// Определение структуры для очереди

typedef struct Queue {

Node\* front; // первый элемент

Node\* rear; // последний

} Queue;

// Функция для создания очереди

Queue\* createQueue() {

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

if (queue == NULL) {

printf("Ошибка выделения памяти для очереди!\n");

exit(1);

}

queue->front = queue->rear = NULL;

return queue;

}

// Функция для добавления элемента в очередь

void enqueue(Queue\* queue, int value) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (newNode == NULL) {

printf("Ошибка выделения памяти для нового элемента!\n");

exit(1);

}

newNode->inf = value;

newNode->next = NULL;

if (queue->rear == NULL) {

queue->front = queue->rear = newNode;

return;

}

queue->rear->next = newNode;

queue->rear = newNode;

}

// Функция для извлечения элемента из очереди

int dequeue(Queue\* queue) {

if (queue->front == NULL) {

printf("Очередь пуста!\n");

exit(1);

}

Node\* temp = queue->front;

int value = temp->inf;

queue->front = temp->next;

if (queue->front == NULL) {

queue->rear = NULL;

}

free(temp);

return value;

}

void addSp(Node\*\* arr\_sp, int\*\* arr, int rows) {

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

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

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

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->inf = j + 1;

newNode->next = arr\_sp[i];

arr\_sp[i] = newNode;

}

}

}

}

void printSp(Node\*\* arr\_sp, int rows) {

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

printf("Вершина %d смежна с: ", i + 1);

Node\* current = arr\_sp[i];

while (current != NULL) {

printf("%d ", current->inf);

current = current->next;

}

printf("\n");

}

}

void crossSp(Node\*\* arr\_sp, int\* arr\_visited, int rows1) {

Queue\* queue = createQueue();

arr\_visited[rows1 - 1] = 1; // Отмечаем текущую вершину как посещенную

enqueue(queue, rows1);

printf("Результат обхода графа в ширину: ");

while (queue->front != NULL) {

int currentRows = dequeue(queue);

printf("%d ", currentRows);

Node\* current = arr\_sp[currentRows - 1];

while (current != NULL) {

int temp = current->inf;

if (!arr\_visited[temp - 1]) {

arr\_visited[temp - 1] = 1;

enqueue(queue, temp);

}

current = current->next;

}

}

free(arr\_visited);

free(queue);

}

void cross(int\*\* arr, int\* arr\_visited, int rows, int rows1) {

Queue\* queue = createQueue();

arr\_visited[rows1 - 1] = 1;

enqueue(queue, rows1);

printf("Результат обхода графа в ширину: ");

while (queue->front != NULL) {

int current = dequeue(queue);

printf("%d ", current);

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

if (arr[current - 1][i] == 1 && !arr\_visited[i]) {

arr\_visited[i] = 1;

enqueue(queue, i + 1);

}

}

}

free(arr\_visited);

free(queue);

}

void crossQue(int\*\* arr, int\* arr\_visited, int rows, int rows1) {

std::queue<int> q; // пустая очередь

arr\_visited[rows1 - 1] = 1;

q.push(rows1);

while (!q.empty()) {

int current = q.front();

q.pop();

printf("%d ", current);

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

if (arr[current - 1][i] == 1 && !arr\_visited[i]) {

arr\_visited[i] = 1;

q.push(i + 1);

}

}

}

}

void crossSpQue(Node\*\* arr\_sp, int\* arr\_visited, int rows1) {

std::queue<int> q;

arr\_visited[rows1 - 1] = 1;

q.push(rows1);

printf("Результат обхода графа в ширину: ");

while (!q.empty()) {

int currentRows = q.front();

q.pop();

printf("%d ", currentRows);

Node\* current = arr\_sp[currentRows - 1];

while (current != NULL) {

int temp = current->inf;

if (!arr\_visited[temp - 1]) {

arr\_visited[temp - 1] = 1;

q.push(temp);

}

current = current->next;

}

}

}

void main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int\*\* arr = 0; // матрица смежности

int\* arr\_visited = 0; // матрица посещённых вершин

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

double time\_spent = 0.0;

int i, j, rows;

printf("Введите количество вершин графа: ");

scanf("%d", &rows);

Node\*\* arr\_sp = NULL;

// Выделение памяти для массива списков смежности

arr\_sp = (Node\*\*)malloc(rows \* sizeof(Node\*));

if (arr\_sp == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows; i++) {

arr\_sp[i] = NULL;

}

arr = (int\*\*)malloc(rows \* sizeof(int\*));

if (arr == NULL) {

printf("Не удалось выделить память!\n");

return;

}

for (i = 0; i < rows; i++) {

arr[i] = (int\*)malloc(rows \* sizeof(int));

}

arr\_visited = (int\*)malloc(rows \* sizeof(int));

for (i = 0; i < rows; i++) {

arr\_visited[i] = 0; // Инициализируем все вершины как непосещенные

}

// генерировать случайные значения для матрицы смежности

for (i = 0; i < rows; i++) {

for (j = i; j < rows; j++) {

if (i == j) {

arr[i][j] = 0; // на главной диагонали нули

}

else {

arr[i][j] = rand() % 2; // случайные значения 0 или 1

arr[j][i] = arr[i][j]; // симметрично заполнять значения для неориентированного графа

}

}

}

// выводить матрицу смежности на экран

printf("Матрица смежности для графа:\n");

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

printf("%-2d ", arr[i][j]);

}

printf("\n");

}

printf("Списки смежности для графа:\n");

addSp(arr\_sp, arr, rows);

printSp(arr\_sp, rows);

int choice, rows1 = 0;

do {

printf("1. Просмотреть массив\n");

printf("2. Обход графа в ширину с помощью списков\n");

printf("3. Обход графа в ширину с помощью queue\n");

printf("4. Обход графа в ширину с помощью queue и списков\n");

printf("5. Обход графа в ширину\n");

printf("6. Выход\n");

printf("Выберите действие: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Матрица смежности для графа:\n");

for (i = 0; i < rows; i++) {

for (j = 0; j < rows; j++) {

printf("%-2d ", arr[i][j]);

}

printf("\n");

}

printSp(arr\_sp, rows);

break;

case 2:

arr\_visited = (int\*)malloc(rows \* sizeof(int));

for (i = 0; i < rows; i++) {

arr\_visited[i] = 0; // Инициализируем все вершины как непосещенные

}

printf("Введите вершину с которой хотите начать обход: ");

scanf("%d", &rows1);

if (rows1 < 1 || rows1 > rows) {

printf("Некорректный номер вершины!\n");

break;

}

start = clock();

crossSp(arr\_sp, arr\_visited, rows1);

end = clock();

printf("\n");

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

printf("%f\n", time\_spent);

break;

case 3:

arr\_visited = (int\*)malloc(rows \* sizeof(int));

for (i = 0; i < rows; i++) {

arr\_visited[i] = 0; // Инициализируем все вершины как непосещенные

}

printf("Введите вершину с которой хотите начать обход: ");

scanf("%d", &rows1);

if (rows1 < 1 || rows1 > rows) {

printf("Некорректный номер вершины!\n");

break;

}

printf("Результат обхода графа в ширину: ");

start = clock();

crossQue(arr, arr\_visited, rows, rows1);

end = clock();

printf("\n");

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

printf("%f\n", time\_spent);

break;

case 4:

arr\_visited = (int\*)malloc(rows \* sizeof(int));

for (i = 0; i < rows; i++) {

arr\_visited[i] = 0; // Инициализируем все вершины как непосещенные

}

printf("Введите вершину с которой хотите начать обход: ");

scanf("%d", &rows1);

if (rows1 < 1 || rows1 > rows) {

printf("Некорректный номер вершины!\n");

break;

}

start = clock();

crossSpQue(arr\_sp, arr\_visited, rows1);

end = clock();

printf("\n");

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

printf("%f\n", time\_spent);

break;

case 5:

arr\_visited = (int\*)malloc(rows \* sizeof(int));

for (i = 0; i < rows; i++) {

arr\_visited[i] = 0; // Инициализируем все вершины как непосещенные

}

printf("Введите вершину с которой хотите начать обход: ");

scanf("%d", &rows1);

if (rows1 < 1 || rows1 > rows) {

printf("Некорректный номер вершины!\n");

break;

}

start = clock();

cross(arr, arr\_visited, rows, rows1);

end = clock();

printf("\n");

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

printf("%f\n", time\_spent);

break;

case 6:

system("cls");

printf("До свидания!\n");

break;

default:

printf("Некорректный выбор!\n");

break;

}

} while (choice != 6);

// Освобождение памяти

for (i = 0; i < rows; i++) {

free(arr[i]);

}

free(arr);

}