Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма обхода графа в ширину»

Выполнил: Швырев А.И.

студент группы 23ВВВ1

Принял: Митрохин М.А.

Пенза 2024

**Содержание**

[Реферат](#_TOC_250009) 5

[Введение](#_TOC_250008) 6

[Постановка задачи](#_TOC_250007) 7

[Описание алгоритма программы](#_TOC_250005) 8

[Описание программы 9](#_TOC_250004)

[Тестирование](#_TOC_250003) 15

[Ручной расчёт задачи](#_TOC_250002) 17

[Заключение](#_TOC_250001) 18

[Список литературы](#_TOC_250000) 19

Приложение A. Листинг программы 20

# Реферат

Отчет 33 стр, 12 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОБХОД ГРАФА В ШИРИНУ.

Целью данного исследования является разработка программы, способной выполнять обход графа в ширину для анализа компонент связности орграфа.

Граф — это абстрактная структура, состоящая из вершин и рёбер, соединяющих эти вершины. В теории графов важным аспектом является понимание связности графа, которая определяет, насколько хорошо вершины графа связаны между собой.

Компоненты связности

Компоненты связности графа — это максимальные подмножества вершин, в которых каждая пара вершин соединена путем. Для ориентированных графов также существует понятие компонент сильной связности, где каждая вершина достижима из любой другой в пределах компоненты.

# Введение

Поиск в ширину (BFS)

Поиск в ширину (BFS) — это алгоритм, который позволяет исследовать все вершины графа, начиная с заданной начальной вершины. Он работает следующим образом:

1. Начните с добавления начальной вершины в очередь.

2. Пока очередь не пуста:

- Извлеките вершину из очереди.

- Посетите все её соседние вершины, добавляя их в очередь, если они ещё не были посещены.

Этот алгоритм гарантирует, что каждая вершина будет посещена ровно один раз и позволяет находить кратчайшие пути в несвязных и слабо связанных графах.

Применение BFS для выделения компонент связности

С помощью алгоритма поиска в ширину можно выделить компоненты связности в графе. Основные шаги включают:

1. Инициализация массива для отслеживания посещённых вершин.

2. Запуск BFS для каждой непосещённой вершины, что приведет к выделению всех вершин, принадлежащих к одной компоненте связности.

3. Повторение процесса до тех пор, пока все вершины не будут посещены.

В данной работе рассматривается применение алгоритма поиска в ширину для выделения компонент связности в графах. Это важно для анализа структуры данных и оптимизации различных процессов в компьютерных науках. Использование языка программирования C и среды разработки Microsoft Visual Studio 2022 позволяет реализовать данный алгоритм эффективно и наглядно.

# 

# Постановка задачи

Требуется разработать программу, которая выделит компоненты связности орграфа, используя алгоритм **поиска в ширину** (BFS). Исходный граф в программе должен задаваться матрицей смежности, при этом при генерации данных должны быть предусмотрены граничные условия.

Основные требования к программе:

1. Ввод данных: Программа должна позволять пользователю вводить количество вершин для генерации матрицы смежности.

2. Генерация матрицы смежности: На основе введённого количества вершин должна быть сгенерирована матрица смежности орграфа.

3. Вывод информации: После обработки данных на экран должны выводиться:

- Матрица смежности орграфа.

- Все компоненты связности орграфа.

4. Обработка ошибок: Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала корректно в любых ситуациях.

Устройство ввода

Программа должна поддерживать ввод данных с использованием клавиатуры и мыши.

Таким образом, разработанная программа будет эффективно выполнять функцию обхода графа в ширину, позволяя пользователю анализировать структуру орграфа и его компоненты связн

# Описание алгоритма программы

Для реализации алгоритма обхода графа в ширину (BFS) в данной программе используются несколько массивов и функций. Основные элементы программы включают создание матрицы смежности, выполнение обхода графа и вывод результатов. Ниже представлено подробное описание каждого компонента алгоритма.

Основные структуры данных

1. Матрица смежности (int\*\* G):

- Используется для хранения информации о рёбрах графа. Если существует ребро между вершинами i и j, то элемент G[i][j] будет равен 1, в противном случае — 0.

2. Массив посещённых вершин (int\* vis):

- Хранит информацию о том, были ли посещены вершины во время обхода. Элемент vis[i] равен 1, если вершина i была посещена, и 0 в противном случае.

Процесс создания графа

Функция createG(int size) отвечает за генерацию матрицы смежности для орграфа:

- Выделяет память под матрицу размером size x size.

- Заполняет матрицу случайными значениями (0 или 1), при этом исключая самосоединения (G[i][i] = 0).

- Обеспечивает симметричность матрицы для неориентированного графа, устанавливая G[j][i] равным G[i][j].

Вывод графа

Функция printG(int\*\* G, int size) выводит на экран матрицу смежности:

- Проходит по всем элементам матрицы и печатает значения, представляя структуру графа.

Алгоритм обхода в ширину (BFS)

Функция DFS(int\*\* G, int size, int\* vis, int s) реализует алгоритм обхода в ширину:

1. Инициализация:

- Создаётся очередь для хранения вершин, которые необходимо посетить.

- Начальная вершина s добавляется в очередь и помечается как посещённая (vis[s] = 1).

2. Обход графа:

- В цикле выполняется следующее:

- Извлекается вершина из начала очереди.

- Печатается номер текущей вершины.

- Для каждой соседней вершины проверяется наличие ребра (G[s][i] == 1) и её статус посещения (vis[i] == 0). Если обе проверки положительны, вершина добавляется в очередь и помечается как посещённая.

Основная функция программы

В функции main() происходит:

1. Ввод данных:

- Пользователь вводит количество вершин для генерации матрицы смежности.

- Генерируется матрица с помощью функции createG().

- Матрица выводится на экран с помощью printG().

2. Запуск обхода:

- Пользователь вводит начальную вершину для запуска обхода.

- Выполняется вызов функции DFS() для выполнения обхода в ширину.

3. Завершение работы:

- Программа завершает свою работу после выполнения обхода и вывода результатов.

Таким образом, программа реализует алгоритм обхода в ширину для орграфа, позволяя пользователю вводить данные и получать результаты в виде матрицы смежности и последовательности посещённых вершин.

# Описание программы

Для написания данной программы использован язык программирования Си. Язык программирования Си - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Работа программы начинается с запроса у пользователя на ввод количества вершин для генерации матрицы смежности орграфа. После того как пользователь введёт это значение, программа создаёт матрицу смежности, представляющую граф, с помощью функции createG(size), которая случайным образом устанавливает наличие или отсутствие рёбер между вершинами, исключая самосоединения. Затем программа выводит сгенерированную матрицу на экран с помощью функции printG(G, size), позволяя пользователю увидеть структуру графа, и предлагает ввести начальную вершину для начала обхода. Программа выделяет память для массива vis, который отслеживает посещённые вершины, и инициализирует его нулями. После ввода начальной вершины запускается функция DFS(G, size, vis, s), выполняющая обход графа в ширину, начиная с указанной вершины. Таким образом, программа позволяет пользователю генерировать орграф с заданным количеством вершин и затем выполнять обход этого графа, отображая последовательность посещённых вершин.

1. Создание матрицы смежности

int\*\* createG(int size) {

int\*\* G = NULL;

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

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

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

}

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

for (int j = i; j < size; j++) {

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

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

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

}

}

return G;

}

1.Инициализация графа: Создаётся указатель на указатель `G`, который будет представлять матрицу смежности.

2.Выделение памяти: С помощью `malloc` выделяется память для массива указателей на строки матрицы.

3.Заполнение матрицы:

- Внешний цикл проходит по всем вершинам `i`.

- Внутренний цикл заполняет верхнюю половину матрицы случайными значениями (0 или 1), представляя наличие рёбер между вершинами.

- Если `i` равно `j`, устанавливается значение 0, чтобы исключить самосоединения.

- Симметричность матрицы обеспечивается присваиванием значения `G[j][i]` равным `G[i][j]`.

2. Вывод матрицы смежности

void printG(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

}

1.Циклы для вывода: Два вложенных цикла проходят по всем элементам матрицы.

2.Печать элементов: Каждый элемент матрицы выводится на экран, формируя визуальное представление графа.

3. Алгоритм обхода в ширину (BFS)

void DFS(int\*\* G, int size, int\* vis, int s) {

queue <int> q;

q.push(s);

vis[s] = 1;

while (!q.empty()) {

s = q.front();

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

q.pop();

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

if (G[s][i] == 1 && vis[i] == 0) {

q.push(i);

vis[i] = 1;

}

}

}

}

1.Инициализация очереди: Создаётся очередь `q`, в которую помещается начальная вершина `s`.

2.Пометка посещённой вершины: Вершина `s` помечается как посещённая в массиве `vis`.

3.Цикл обхода:

- Пока очередь не пуста:

- Извлекается вершина из начала очереди и выводится её номер.

- Для каждой соседней вершины проверяется наличие ребра (`G[s][i] == 1`) и её статус посещения (`vis[i] == 0`).

- Если обе проверки пройдены, соседняя вершина добавляется в очередь и помечается как посещённая.

4. Основная функция программы

int main() {

int\*\* G = NULL;

int\* vis = NULL;

int size = 5;

int s = 1;

printf("Введите колл-во вершин: ");

scanf\_s("%d", &size);

G = createG(size);

printG(G, size);

printf("Введите начальную вершину: ");

vis = (int\*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

{

vis[i] = 0;

}

scanf\_s("%d", &s);

s--;

printf("Обход графа в ширину: \n");

DFS(G, size, vis, s);

scanf\_s("%d", s);

return 0;

}

1.Инициализация переменных: Объявляются указатели для графа и массива посещённых вершин, а также переменные для хранения размера графа и начальной вершины.

2.Ввод данных от пользователя:

- Пользователь вводит количество вершин для генерации графа.

- Генерируется матрица смежности с помощью функции `createG()`.

- Матрица выводится на экран с помощью функции `printG()`.

3.Запуск обхода:

- Пользователь вводит начальную вершину для запуска обхода.

- Выделяется память для массива посещённых вершин и инициализируется нулями.

- Выполняется вызов функции `DFS()` для выполнения обхода в ширину, начиная с указанной вершины.

4.Завершение работы программы: Программа завершает свою работу после выполнения обхода и вывода результатов.

Таким образом, программа реализует алгоритм обхода в ширину для орграфа, позволяя пользователю вводить данные и получать результаты в виде матрицы смежности и последовательности посещённых вершин.

# Тестирование

Среда разработки MicrosoftVisualStudio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования не было выявлено проблем, связанных с вводом данных, выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и начальной вершины.

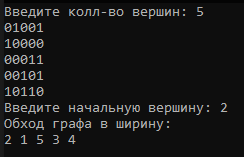


Рисунок 1 – Тестирование при вводе количество вершин = 5 и начальная вершина = 2

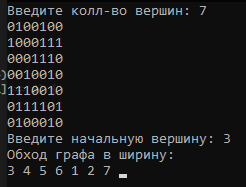


Рисунок 2 – Тестирование при вводе количество вершин = 7 и начальная вершина = 3

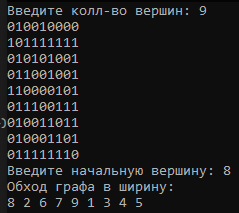


Рисунок 3 – Тестирование при вводе количество вершин = 9 и начальная вершина = 8

## Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере графа с 5 вершинами (рисунок 1).

За начальную вершину была выбрана 2-ая. Начинаем обход из 2 вершины в последующие. Проверяем, если есть путь из 0 в другие вершины, то дальше идем. В нашем случае идем в вершину 1. Аналогично проверяем и в вершине 1. Из вершины 1 идем в 2. Из вершины 2 нет путей, которые нам не известны, поэтому возвращаемся в 1. Из 1 вершины идём в 5. Из вершины 5 идём в 1 вершину. Из вершины 1 нет путей, которые нам не известны, поэтому возвращаемся в 5. Возвращаемся в 5 и проверяем. Из 5 есть путь в 3, идем туда. Из 3 идем в 4 вершину. Из 4 вершины есть путь в 3. В 3 вершине нет неизвестных нам путей, поэтому возвращаемся в предыдущие и проверяем. Проверка показала, что все вершины пройдены. Получается обход графа в ширину равен: {2, 1, 5, 3, 4}

Программа показала такие же значения.

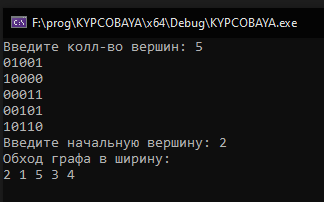


Рисунок 4 - Тестирование работы программы

Таким образом, можно сделать вывод, что программа работает верно.

# Заключение

Таким образом, в процессе создания данного проекта была разработана программа, реализующая алгоритм обхода графа в ширину для поиска компонент связности орграфа с использованием языка программирования C++ в среде Microsoft Visual Studio 2022. В ходе выполнения курсовой работы были приобретены навыки разработки программного обеспечения и освоены методы создания матриц смежности, основанных на теории графов. Также были углублены знания по алгоритму обхода в ширину, что позволило лучше понять его применение в задачах анализа графов. Однако недостатком разработанной программы является примитивный пользовательский интерфейс, так как программа работает в консольном режиме, что упрощает реализацию по сравнению с более сложными оконными интерфейсами. Тем не менее, программа обладает достаточным функционалом для эффективного использования, позволяя пользователям генерировать орграфы и выполнять их анализ с помощью обхода в ширину.

# Список литературы

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с.
2. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
3. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
4. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.
5. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.
6. 3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.

## Приложение

**Листинг программы.**

#include<iostream>

#include <queue>

#include <cstdlib>

using namespace std;

int\*\* createG(int size) {

int\*\* G = NULL;

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

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

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

}

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

for (int j = i; j < size; j++) {

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

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

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

}

}

return G;

}

void printG(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

}

void DFS(int\*\* G, int size, int\* vis, int s) {

queue <int> q;

q.push(s);

vis[s] = 1;

while (!q.empty()) {

s = q.front();

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

q.pop();

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

if (G[s][i] == 1 && vis[i] == 0) {

q.push(i);

vis[i] = 1;

}

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int\*\* G = NULL;

int\* vis = NULL;

int size = 5;

int s = 1;

printf("Введите колл-во вершин: ");

scanf\_s("%d", &size);

G = createG(size);

printG(G, size);

printf("Введите начальную вершину: ");

vis = (int\*)malloc(size \* sizeof(int));

for (int i = 0; i < size; i++)

{

vis[i] = 0;

}

scanf\_s("%d", &s);

s--;

printf("Обход графа в ширину: \n");

DFS(G, size, vis, s);

scanf\_s("%d", s);

return 0;

}