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 нахождения гамильтоновых циклов»

Выполнил студент группы 20ВВ3:

Мальков И.А.

Приняли:

Митрохин М.А.

Юрова О.В.

Пенза 2021

Оглавление

[**Введение** 5](#_Toc90283082)

[**1.** **Постановка задачи** 6](#_Toc90283083)

[**2.** **Теоретическая часть задания** 7](#_Toc90283084)

[**3.** **Описание алгоритма поставленной задачи** 8](#_Toc90283085)

[**4.** **Пример ручного расчёта задачи и вычислений** 9](#_Toc90283086)

[**5.** **Описание самой программы** 11](#_Toc90283087)

[**6.** **Тесты** 14](#_Toc90283088)

[**Заключение. Вывод о работе программы** 15](#_Toc90283089)

[**Список использованных источников** 16](#_Toc90283090)

[**Приложение А. Листинг**](#_Toc90283091) [**программы.** 17](#_Toc90283091)

**Введение**

Теория графов стала активно применяться в программировании одновременно с использованием ЭВМ в силу удобного выражения задач обработки информации на теоретико-графовом языке. Модель программы в виде управляющего графа, модель арифметического выражения в виде ориентированного дерева, синтаксические деревья, деревья сортировки, сети Петри и другие теоретико-графовые конструкции внесли свой существенный вклад в развитие программирования и его автоматизации. Появление суперкомпьютеров и сетей и возникшая при этом проблема эффективной организации параллельных и распределенных вычислений над информационными массивами большого объема подтвердили тенденцию использования графов как наиболее эффективного средства автоматизации программирования.

С точки зрения компьютерных наук и дискретной математики, графы  — это абстрактный способ представления типов отношений, например дорог, соединяющих города, и других видов сетей. Графы состоят из рёбер и вершин. Вершина — это точка на графе, а ребро — это то, что соединяет две точки на графе.

Основные алгоритмы на графах – это обход графа и нахождение кратчайшего пути. Есть два пути обхода вершин в графе: поиск в ширину и глубину.

1. **Постановка задачи**

Требуется разработать программу, которая будет находить гамильтонов цикл в графе.

Исходный граф в программе должен задаваться матрицей смежности. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

В качестве среды разработки использовать редактор исходного кода Visual Studio Code, язык программирования – С++, операционную система - Linux.

Необходимо разработать программу с графическим интерфейсом, которая будет находить гамильтонов цикл в матрице смежности. Матрица, на выбор пользователя создается либо автоматически, либо заполняется вручную. После создания матрицы необходимо предусмотреть сохранение ее в текстовый файл для дальнейшего считывания и анализа. Результат работы программы также сохранить в текстовый файл.

1. **Теоретическая часть задания**

Гамильтонов путь в графе - путь, проходящий через каждую вершину графа в точности один раз.

Гамильтонов цикл в графе - замкнутый гамильтонов путь.

Граф называется Гамильтоновым, если в нем существует простой цикл, содержащий все вершины графа. Например, каждый полный граф – гамильтонов, потому что в нем проведены всевозможные ребра и, в частности, те, благодаря которым возможен обход по всем вершинам. Общих и легко осуществляемых действий, с помощью которых можно было бы достоверно выяснить, является ли данный граф гамильтоновым, не существует. Однако, имеются достаточные условия на гамильтоновость, которые проверяются легко. Недостаток здесь состоит в том, что даже если ни одно из этих условий не выполняется, граф может оказаться гамильтоновым.

Например, условие первое - Условие Дирака. Пусть p - число вершин в данном графе; если степень каждой вершины не меньше, чем p/2 , то граф называется Графом Дирака. Можно доказать, что Каждый граф Дирака обязательно гамильтонов.

Условие второе - Условие Оре. Если для любой пары неcмежных вершин выполнено неравенство d(x) + d(y) >= p, то граф называется Графом Оре (словами: степени любых двух несмежных вершин не меньше общего числа вершин в графе). Можно доказать, что Всякий граф Оре обязательно гамильтонов.

В данной работе для проверки наличия гамильтонова цикла использовалось условие Оре, ввиду большей точности.

В качестве алгоритма нахождения гамильтонова цикла использовался так называемый жадный алгоритм. Он позволяет начать перебор вершин с выбранной пользователем. Далее осуществляется проход по смежным вершинам графа, помечаются посещенные вершины. Концом алгоритма является достижение точки входа, выбранной вершины.

1. **Описание алгоритма поставленной задачи**

В самом начале функции поиска гамильтонова цикла переменной q, отвечающей за наличие пути, присваивается значение 0 (путь не найден). Далее запускается цикл обхода матрицы по всем вершинам, с условием пока путь не найден. В цикле идет условие проверки наличия ребра между текущей вершиной и вершиной, найденной при предыдущем вызове. В этом условии проверяется пройдены ли все вершины и достигнута ли начальная. Если да, то переменной q присваивается 1 (путь найден). Если нет, то формируется путь, при условии, что текущая вершина v еще не была посещена. Далее сохраняется номер прохода и текущая вершина добавляется к найденному пути. После этого функция рекурсивно вызывается для анализа следующей вершины. Если путь не найден, то текущая вершина помечается как непосещенная. В случае, если текущая вершина v уже была посещена, то программа переходит на анализ следующей вершины. В конце функция возвращает найден путь или нет.

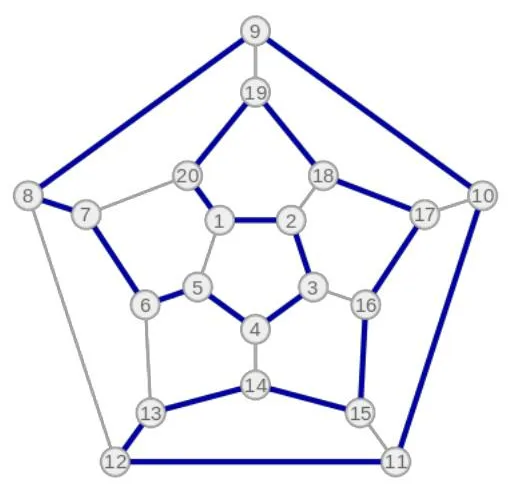


Рисунок 1 - Пример гамильтонова цикла

Алгоритм gamilton

1. V = 0;
2. Q = 0;
3. Для всех v, если путь не найден выполнять
4. Если есть ли ребро между G1[v][path[k - 1]] и G1[path[k - 1]][v]
5. То
6. {
7. Если k == n и v == v0
8. То путь найден;
9. Иначе
10. {
11. Если c[v] == -1
12. То
13. {
14. Помечаем текущую как посещенную;
15. Добавляем вершину в найденный путь;
16. Q = gamilton(k + 1, path, c);
17. Если путь не найден
18. То помечаем текущую как не посещенную
19. }
20. }
21. Иначе continue;
22. }
23. Возвращаем q;
24. **Пример ручного расчёта задачи и вычислений**

Проведу проверку алгоритма поиска гамильтонова цикла на данной матрице смежности.



Рисунок 2 - Матрица смежности

На этапе подготовки выделяется память для массива непосещенных вершин и массива пути. Матрица вершин заполняется -1, в матрице пути на первое место записывается вершина, с которой начинаем. Например, 1. В матрице непосещенных вершин она отмечается как посещенная.



Рисунок 3 - Массив вершин



Рисунок 4 - Массив пути

Далее программа находит ребро между вершиной 0 и вершиной 1. Добавляет 0 к массиву пути и делает его посещенным.



Рисунок 5 - Массив вершин



Рисунок 6 - Массив пути

Теперь цикл проходит по вершине 0 и находит связь с непосещенной вершиной 2. Добавляет 2 к пути, отмечает посещенной и ищет дальнейшую связь.



Рисунок 7 - Массив вершин



Рисунок 8 - Массив пути

При анализе вершины 2 находится связь с вершиной 3. Дальше она так же анализируется, добавляется к пути и отмечается помеченной.



Рисунок 9 - Массив вершин



Рисунок 10 - Массив пути

Далее 3 связана из непосещенных вершин только с 4. Она добавляется к пути и в свою очередь становится посещенной.



Рисунок 11 - Массив вершин



Рисунок 12 - Массив пути

Путь найден, в конце дописывается начальная вершина, алгоритм выполнен.



Рисунок 12 - Массив пути

Рассчитанный результат сходится с результатом тестового запуска.

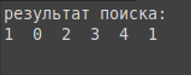


Рисунок 13 — Результат тестового запуска

1. **Описание самой программы**

Алгоритм поиска гамильтонова цикла находится в функции gamilton(). В эту функцию передаются такие параметры, как номер прохода и заранее созданные массивы пути и непосещенных вершин. Переменная k – номер прохода, path[ ] – массив пути, c[ ] – массив непосещенных вершин. Переменная v отвечает за индекс вершины, далее по ней будет проходить цикл. Переменная q может иметь значения 0 или 1, что означает цикл не найден или цикл найден соответственно. После определения этих переменных начинается цикл по v до n, где n – количество вершин, с условием, что путь не найден, то есть !q. for (v = 0; v < n && !q; v++) – так выглядит цикл. Сначала в цикле проверяется условие существования ребра между текущей вершиной и вершиной, найденной при предыдущем вызове. if (G1[v][path[k - 1]] || G1[path[k - 1]][v]) – так выглядит условие. G1[ ][ ] – анализируемая матрица. Если ребро найдено, то проверяется условие if (k == n && v == v0), которое означает “если обошли все вершины и дошли до начальной”. Переменная v0 – вершина, с которой начинается обход, она задается пользователем. Если условие выполнилось, то q присваивается 1, значит путь найден, программа возвращает q, выходит из функции. Если нет, то формируется путь, если в вершине v еще не были. То есть проверяется условие if (c[v] == -1) и если оно истинно, то c[v] присваивается номер прохода k, текущая вершина добавляется к массиву пути по индексу k (path[k] = v;). После этого для q рекурсивно вызывается эта функция для анализа следующей вершины и если путь не найден, то текущую вершину помечаем как непосещенную.

1. **Тесты**

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

* 1. Тестирование программы при автоматическом заполнении матрицы. Ожидается нормальное завершение.

Выбран режим автозаполнения и создана матрица. Данные сохранены и записаны в файл.

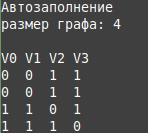


Рисунок 14 — Исходная матрица

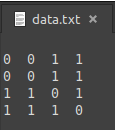


Рисунок 15 — Результат сохранения

Произведен поиск цикла, начиная с вершины 1. Результат сохранен и записан в файл.

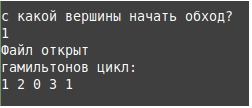


Рисунок 16 — Результат выполнения

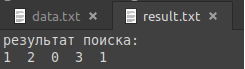


Рисунок 17 — Результат сохранения

* 1. Тестирование программы при ручном заполнении матрицы.

Выбран режим ручного заполнение и создана матрица. Ожидается нормальное завершение. Данные сохранены и записаны в файл.

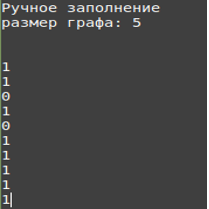


Рисунок 18 — Заполнение матрицы

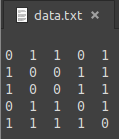


Рисунок 19 — Результат сохранения

Произведен поиск цикла, начиная с вершины 1. Результат сохранен и записан в файл.

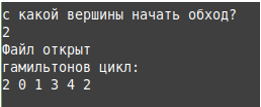


Рисунок 20 — Результат выполнения

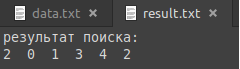


Рисунок 21 — Результат сохранения

* 1. Тестирование на матрице, где не может быть гамильтонова цикла. Ожидается вывод предупреждения.

Создана матрица, где нет гамильтонова цикла.

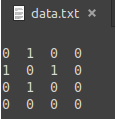


Рисунок 22 — Исходная матрица

При попытке выполнить программу, результатом функции проверки матрицы является вывод предупреждения и программа не зайдет в функцию поиска цикла.

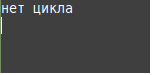


Рисунок 23 — Вывод предупреждения

**Заключение. Вывод о работе программы**

При выполнении данного курсового проекта, был получен опыт разработки программ в Visual Studio Code, освоен алгоритм поиска гамильтонова цикла в графе и углублены знания языка C++. Так же был приобретен опыт разработки, отладки и запуска программ в терминале операционной системы Linux.

Недостатком данной программы является то, что она не является кроссплатформенной и написана с использованием средств, доступных только в ОС Linux. Но также она имеет достаточный функционал для выполнения поставленной задачи и большой потенциал для дальнейшего ее совершенствования.

**Список использованных источников**

https://habr.com/ru/post/65367/

https://www.iis.nsk.su/files/articles/sbor\_kas\_07\_kasyanov\_primenenie.pdf

https://gist.github.com/sampletext32/98d870dd76c0b4e4187a6f5c07b7784d#16

https://habr.com/ru/post/160077/

https://medium.com/nuances-of-programming/графы-основы-теории-алгоритмы-поиска-b93672f59747

**Приложение А. Листинг программы**

Файл header.hpp

#include <stdio.h>

#include <termios.h>

#include <unistd.h>

#include <stdlib.h>

#include <iostream>

#include <cstring>

#include <time.h>

#include <fstream>

using namespace std;

int main();

int mygetch();

void prepare(int n);

void check();

void save\_data(int n);

void save\_res();

int gamilton (int k);

void printGam();

void printGam\_s();

int\*\* create(int n);

int\*\* m\_create(int n);

void size\_A();

void size\_M();

void pos(int a);

int mainmenu();

Файл main.cpp

#include "header.hpp"

int n, er;

int v0;

int \*\*t;

int \*\*G;

int \*\*G1;

int a = 0;

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

enum

{

KEY\_UP = 65,

KEY\_DOWN = 66,

ENTER = 10

};

int mygetch(){

struct termios oldt,newt;

int ch;

tcgetattr( STDIN\_FILENO, &oldt );

newt = oldt;

newt.c\_lflag &= ~( ICANON | ECHO );

tcsetattr( STDIN\_FILENO, TCSANOW, &newt );

ch = cin.get();

tcsetattr( STDIN\_FILENO, TCSANOW, &oldt );

return ch;

}

void pos(int a) {

if (a == 0) {

system("clear");

cout <<

" \n"

" / \\\n"

" | >> РУЧНОЕ ЗАПОЛНЕНИЕ << |\n"

" \\ /\n"

" \n"

" / \\\n"

" | автозаполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранение данных |\n"

" \\ /\n"

" \n"

" / \\\n"

" | поиск гамильтонова цикла |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранить результат |\n"

" \\ /\n"

" \n"

" / \\\n"

" | выход |\n"

" \\ /\n";

}

if (a == 1) {

system("clear");

cout <<

" \n"

" / \\\n"

" | ручное заполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | >> АВТОЗАПОЛНЕНИЕ << |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранение данных |\n"

" \\ /\n"

" \n"

" / \\\n"

" | поиск гамильтонова цикла |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранить результат |\n"

" \\ /\n"

" \n"

" / \\\n"

" | выход |\n"

" \\ /\n";

}

if (a == 2) {

system("clear");

cout <<

" \n"

" / \\\n"

" | ручное заполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | автозаполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | >> СОХРАНЕНИЕ ДАННЫХ << |\n"

" \\ /\n"

" \n"

" / \\\n"

" | поиск гамильтонова цикла |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранить результат |\n"

" \\ /\n"

" \n"

" / \\\n"

" | выход |\n"

" \\ /\n";

}

if (a == 3) {

system("clear");

cout <<

" \n"

" / \\\n"

" | ручное заполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | автозаполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранение данных |\n"

" \\ /\n"

" \n"

" / \\\n"

" | >> ПОИСК ГАМИЛЬНТОНОВА ЦИКЛА << |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранить результат |\n"

" \\ /\n"

" \n"

" / \\\n"

" | выход |\n"

" \\ /\n";

}

if (a == 4) {

system("clear");

cout <<

" \n"

" / \\\n"

" | ручное заполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | автозаполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранение данных |\n"

" \\ /\n"

" \n"

" / \\\n"

" | поиск гамильтонова цикла |\n"

" \\ /\n"

" \n"

" / \\\n"

" | >> СОХРАНИТЬ РЕЗУЛЬТАТ << |\n"

" \\ /\n"

" \n"

" / \\\n"

" | выход |\n"

" \\ /\n";

}

if (a == 5) {

system("clear");

cout <<

" \n"

" / \\\n"

" | ручное заполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | автозаполнение |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранение данных |\n"

" \\ /\n"

" \n"

" / \\\n"

" | поиск гамильтонова цикла |\n"

" \\ /\n"

" \n"

" / \\\n"

" | сохранить результат |\n"

" \\ /\n"

" \n"

" / \\\n"

" | >> ВЫХОД << |\n"

" \\ /\n";

}

}

int mainmenu () {

int ch;

pos(a);

while(true){

do {

ch = mygetch();

switch(ch){

case KEY\_UP:

if (a == 0){

a = 6;

pos(a);

}

if (a > 0){

a = a - 2;

pos(a);

}

case KEY\_DOWN:

if (a < 6){

a++;

pos(a);

}

if (a == 6){

a = 0;

pos(a);

}

default:

break;

}

} while (ch != ENTER);

if (a == 0)

{

system("clear");

a = -1;

size\_M();

}

if (a == 1)

{

system("clear");

a = -1;

size\_A();

}

if (a == 2)

{

system("clear");

a = -1;

save\_data(n);

}

if (a == 3)

{

system("clear");

a = -1;

check();

}

if (a == 4)

{

system("clear");

a = -1;

save\_res();

}

if (a == 5)

{

system("clear");

exit(3);

}

}

return 0;

}

void size\_A(){

n = 0;

cout << "Автозаполнение" << endl;

cout << "размер графа: ";

cin >> n;

cout << endl;

t = create(n);

cout << endl;

mainmenu();

}

void size\_M(){

n = 0;

cout << "Ручное заполнение" << endl;

cout << "размер графа: ";

cin >> n;

cout << endl;

t = m\_create(n);

cout << endl;

pos(0);

}

int\*\* create(int n){

int k = 1;

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

srand(time(NULL));

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

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

G[i][i] = 0;

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

if (rand() % 100 > 50){

G[i][j] = 0;

}

else{

G[i][j] = 1;

}

}

k++;

}

k = 1;

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

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

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

}

k++;

}

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

cout << "V" << i << " ";

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

cout << "\n";

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

cout << G[i][j] << " ";

}

}

return G;

}

int\*\* m\_create(int n){

int k = 1;

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

cout << endl;

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

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

G[i][i] = 0;

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

cin >> G[i][j];

}

k++;

}

k = 1;

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

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

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

}

k++;

}

cout << endl;

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

cout << "V" << i << " ";

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

cout << endl;

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

cout << G[i][j] << " ";

}

}

return G;

}

void printGam(int\* path)

{

int p;

for (p = 0; p < n; p++)

cout << path[p] << " ";

cout << path[0];

cout << endl;

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

res[i] = path[i];

}

int gamilton(int k, int\* path, int\* c){

int v;

int q = 0;

for (v = 0; v < n && !q; v++){

if (G1[v][path[k - 1]] || G1[path[k - 1]][v]){

if (k == n && v == v0)

q = 1;

else

if (c[v] == -1){

c[v] = 1;

path[k] = v;

q = gamilton(k + 1, path, c);

if (!q)

c[v] = -1;

}

else

continue;

}

} return q;

}

void save\_data(int n){

ofstream fout;

fout.open("data.txt");

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

fout << endl;

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

fout << G[i][j] << " ";

}

}

cout << "данные сохранены" << endl;

fout.close();

mainmenu ();

}

void check(){

int a = 0, b = 0, s = 1, l = 1, x = 0;

if (n >= 3){

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

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

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

x++;

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

if (G[i][p] == 1)

a++;

if (G[j][p] == 1)

b++;

}

if (a + b >= n){

prepare(n);

return;

}

}

}

s++;

a = 0; b = 0;

}

l++;

s = l;

}

if (x == 0){

prepare(n);

return;

}

else{

cout << "нет цикла" << endl;

return;

}

}

void prepare(int n){

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

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

for (int j = 0; j < n; j++)

c[j] = -1;

cout << endl << "с какой вершины начать обход?" << endl;

cin >> v0;

path[0] = v0;

c[v0] = v0;

ifstream fin;

fin.open("data.txt");

if (!fin.is\_open())

{

cout << "Ошибка открытия файла!" << endl;

}

else {

cout << "Файл открыт" << endl;

G1 = new int\*[n];

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

G1[i] = new int[n];

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

for (int j = 0; j < n; j++)

fin >> G1[i][j];

}

fin.close();

cout << "гамильтонов цикл:" << endl;

if (gamilton(1, path, c))

printGam(path);

else{

er = 1;

cout << "нет решений" << endl;

}

cout << endl;

mainmenu();

}

void save\_res(){

ofstream fout;

fout.open("result.txt");

fout << "результат поиска:" << endl;

if (er == 1){

fout << "нет решений";

}

else{

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

fout << res[i] << " ";

}

fout << res[0];

}

cout << "результат сохранен" << endl;

fout.close();

mainmenu();

}

int main(){

mainmenu();

}