Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Реализация алгоритма Форда-Фалкерсона для нахождения максимального потока»

Выполнил ст. гр. 22ВВВ2:

Рябов С.И.

Принял:

Митрохин М.А.

Пенза 2023

Содержание

[Реферат 5](#_Toc154672897)

[Введение 6](#_Toc154672898)

[1 Постановка задачи 7](#_Toc154672899)

[2 Выбор решения 8](#_Toc154672900)

[3 Описание алгоритма программы 9](#_Toc154672901)

[4 Описание программы 11](#_Toc154672902)

[5 Отладка 16](#_Toc154672903)

[6 Тестирование 20](#_Toc154672904)

[7 Ручной растёт задачи 22](#_Toc154672905)

[Заключение 24](#_Toc154672906)

[Список используемых источников 25](#_Toc154672907)

[Приложение А (Листинг программы) 26](#_Toc154672908)

# **Реферат**

Отчет 29 страниц, 18 рисунков.

Цель исследования – разработка программы, способной решать задачу нахождения максимального потока в графе, используя алгоритм Форда – Фалкерсона.

В работе рассмотрены принципы работы алгоритма Форда – Фалкерсона для нахождения максимального потока. Установлено, что с помощью данного алгоритма можно найти максимальный поток, исходящий из истока в сток в взвешенном, ориентированном графе.

# **Введение**

Теория графов является важной и фундаментальной областью математики и информатики, которая находит широкое применение в различных сферах науки, техники и компьютерных наук. Графы используются для анализа и моделирования сложных задач, процессов и взаимодействий между объектами.

Графы – универсальный инструмент, играющий значительную роль в разных отраслях человеческой жизни. Они предоставляют набор инструментов и методов для представления и обработки структурированных данных.

Алгоритм Форда-Фалкерсона, который является одним из ключевых алгоритмов теории графов, применяется для решения задач нахождения максимального потока. Это важная задача, которая возникает во многих практических ситуациях, таких как оптимизация транспортной сети, маршрутизация в компьютерных сетях, планирование производства и другие. Понимание и реализация алгоритма Форда-Фалкерсона позволяет осуществлять эффективное решение данных задач, а также анализировать и оптимизировать потоки данных в различных системах.

Таким образом, изучение теории графов и алгоритма Форда-Фалкерсона является важным и интересным для исследования, а также играет большую роль в практическом применении.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2022, язык программирования – С и C++.

Целью данной курсовой работы является разработка программы на языках семейства Cи, которые являются широко используемым. Именно с их помощью в данном курсовом проекте реализуется алгоритм Форда-Фалкерсона для нахождения максимального потока.

# **1 Постановка задачи**

Необходимо разработать программу, которая создает граф, а затем найти наибольший потом при помощи алгоритма Форда-Фалкерсона.

Программа должна создавать взвешенный ориентированный граф, в котором будет исток (вершина графа, из которой только исходят ребра), сток (вершина графа, в которую только входят ребра), а также вершины, которые соединяют исток и сток, каждая из которых имеет минимум одно выходящее и одно входящее ребро.

После заполнения матрицы смежности, надо выполнить обход, чтобы найти путь из истока в сток, а затем из всех ребер вычесть максимальный поток (величина, которая соответствует минимальной пропускной способности из всех вершин, выбранного пути). Путь, а также максимальный поток надо вывести в консоль, затем повторять эти действия, пока путей из истока в сток не останется. В конце программы выводит сумму всех потоков.

# **2 Выбор решения**

Пользователь в начале работы программы выбирает, какой граф хочет создать, ему будут предложены варианты:

* Выбраться размер графа и заполнить его случайно.
* Выбрать заранее заполненный граф.
* Выбрать другой заранее заполненный граф

Для поиска потока будем использовать алгоритм обхода графа в глубину с сохранением пройденного пути в очереди. Если есть путь в сток есть, то алгоритм выбирает его, если нет, то выбирает первый попавшийся. Когда пути нет, то программа удаляет последний элемент из очереди и возвращается к предыдущему обходу.

После того, как обход дошел до стока, программа проходит путь, записанный в очереди, и выбирает наименьшее ребро, вычитает из всех путей значение этого ребра и прибавляет это значение к потоку.

Когда программа в итоге не дойдет то стока, то она завершается и выведет максимальный поток.

# **3 Описание алгоритма программы**

Алгоритм нахождения максимального потока FordFulkersonAlg() для того, чтобы найти путь из истока в сток, использует функцию обхода в глубину. Алгоритм использует матрицу смежности G[][], массив пройденных вершины DIST[] и 2 библиотечные очереди Q, которая служит для хранения пути и Q2, которая нужна для выполнения операций с Q.

При вызове функции, она принимает такие параметры как: начальную вершину (следующую вершину для обхода), матрицу смежности, массив пройденных вершин.

Алгоритм в начале своей работы записывает первую посещенную вершину в список, затем проверяет не достиг ли алгоритм стока. После проверки он запускает в цикл, в котором будет проверено ребро, выходящее в сток и остальные ребра. Если ребро в сток присутствует, то он вызывает рекурсию с вершиной стока, если нет ребра в сток, но есть любое другое, то будет вызвана рекурсия с этой вершиной. Иначе программа увеличивает переменную count, когда она становится равной количеству вершин SizeG или переменная back равна 1, то запускается алгоритм возврата назад, удаляется последняя вершина при помощи функции DelLast() и back становится равна 1.

После завершения работы обхода программа вычитает поток из всех пройденный вершин. Если после цикла переменная count равна количеству вершин SizeG, то путей из истока в сток больше нет, и программу можно завершать.

**FordFulkersonAlg ()**

1.Записать в очередь v

2.**ЕСЛИ** v != SizeG

3. count = 0

4. Для i=0, пока i< SizeG, делать i=i+1

5. Если G[v][ SizeG - 1]>0 и DIST [SizeG - 1] = -1 и check != 1

6. back = 0

7. Вызывть функцию FordFulkersonAlg(SizeG - 1, G, DIST)

8. Конец условия

9. Также если G[v][ ш]>0 и DIST [ш] = -1 и check != 1

10. back = 0

11. Вызвать функцию FordFulkersonAlg(i, G, DIST)

12. Конец условия

13. Иначе

14. count=count+1

15. Если count = SizeG или back = 1 и i = SizeG – 1

16. Вызвать функцию DelLast()

17. back = 1

18. Конец условия

19. Конец условия

20. Конец цикла

21. Если count = SizeG

22. Вернуть 0

22. Конец условия

23. Конец условия

24.Иначе

25. check = 1

26. вернуть 1

27. Конец условия

# **4 Описание программы**

Для написания данной программы использованы языки программирования С++ и C. На сегодняшний день языки семейства Си являются универсальными и пользуются большой популярностью среди программистов, благодаря своим возможностям объединения особенностей языков программирования высокого и низкого уровней.

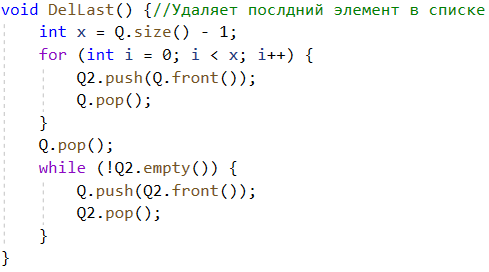


Рисунок 1 - Функция DelLast

Функция DelLast (Рисунок 1) служит для удаления последнего элемента в очереди Q, за счет перезаписи очереди в Q2 x раз, что на 1 элемент меньше, чем хранится в Q.

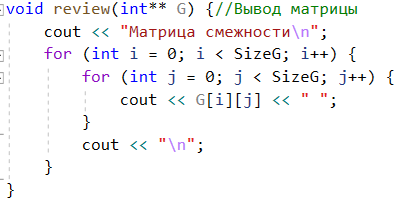


Рисунок 2 - Функция review

Функция review (Рисунок 2) выводит матрицу смежности G на экран.

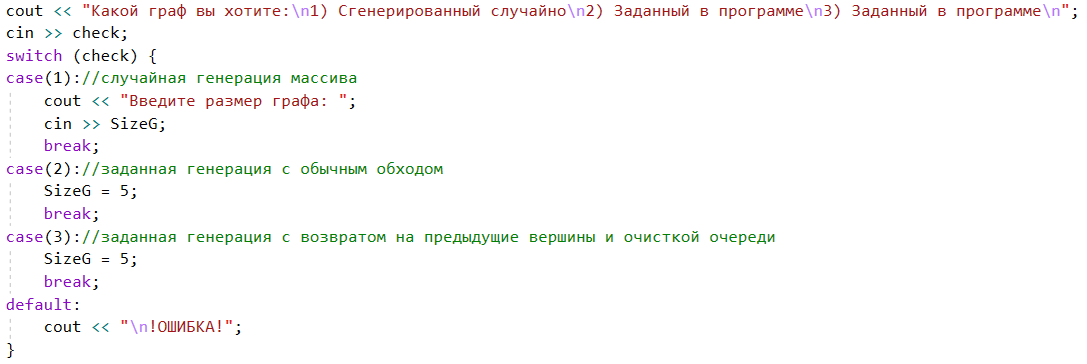


Рисунок 3 - Функция main

Функция main определяет размер графа (Рисунок 3), предлагая пользователю выбрать какой граф он хочет. При вводе 1 пользователю будет предложено самостоятельно выбрать размер графа, при вводе 2 и 3 граф будет фиксированного размера, заданного заранее. Если же будет введено любое другое чисто, то программа выведет надпись «!ОШИБКА!» и потребует ввести значение еще раз.

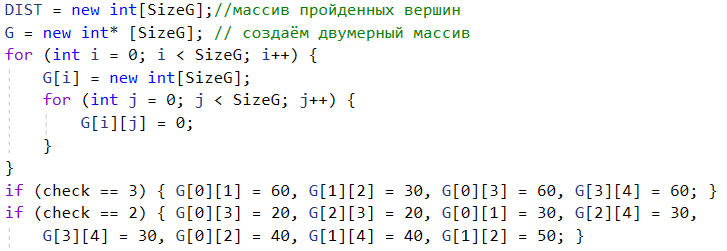


Рисунок 4 - Функция main

Программа создает и инициализирует массив пройденных вершин и матрицу смежностей (Рисунок 4), если пользователь выбрал 2 или 3 пункт, то программа заполняет их заданными числами.

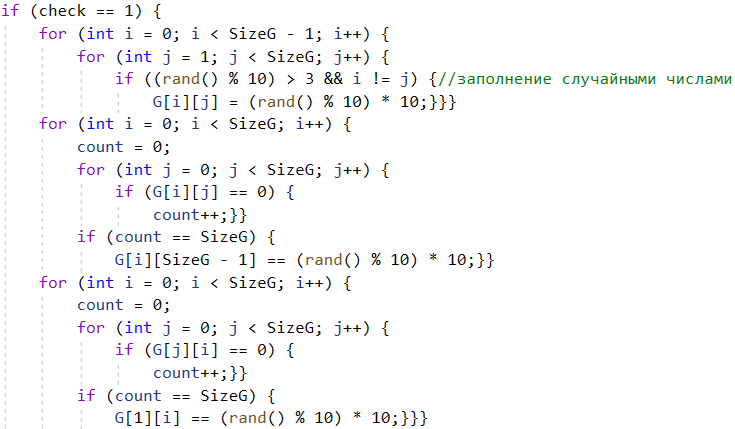


Рисунок 5 - Функция main

Если пользователь выбрал ввести размер вручную, то программа пройдет по всем элементам матрицы, кроме 1 столбика, последней строчки, а также диагонали, и заполнит их случайными числами от 10 до 90 с вероятностью 60% (Рисунок 5). Затем программа проверит наличие стоков и стоков, их должно быть по 1, если их больше, то программа уберет лишние.

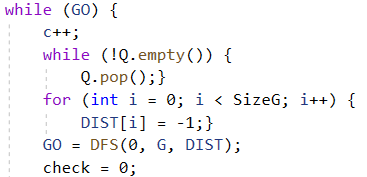


Рисунок 6 - Функция main

Цикл while будет повторяться, пока можно добраться от истока в сток. Программа очищает очередь и массив пройдённых вершин после предыдущего обхода. Запускает алгоритм FordFulkersonAlg. Если не будет достигнут исток, то обход вернёт 0.

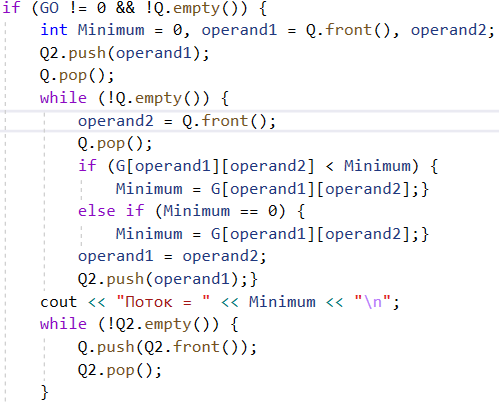


Рисунок 7 - Функция main

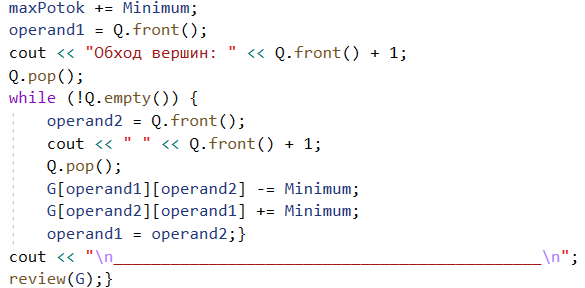


Рисунок 8 - Функция main

После успешного обхода в программе хранится путь, пройденный при обходе. Дальше она проходит его при помощи цикла, перезаписывает значение в другой список и сохраняет наименьшее значение ребра. Затем вычитает из пройденных ребер минимальное значение потока и добавляет его в противоположное направление между этими вершинами (Рисунок 7-8).

В конце программа выводит на экран значение максимального потока.

# **5 Отладка**

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2022. Программа обладает всеми необходимыми средствами для разработки и отладки программы. Для отладки использовались несколько возможностей VisualStudio: Отладка — это процесс поиска и исправления ошибок или неполадок в исходном коде программы. Когда программа не работает или работает не должным образом, нужно использовать отладку. Пользователь при помощи компьютерной отладки исследует код на наличие ошибок.

Пользователь запускает отладку, когда хочет воспользоваться пошаговой проверкой кода, а также проанализировать код и найти проблемы. В разработке программы применялась пошаговая отладка, использовавшаяся для просмотра всего кода и отслеживания ошибок на небольших участках. Также применялась отладка в заданных точках. Внутри программы VisualStudio были расставлены точки остановки для приостановления работы кода в наиболее важных местах. Это нужно для проверки значения переменных и дальнейшего устранения ошибок.

Одно из самых важных мест в программе это обход, а именно проверка смежности с другими вершинами, там стоит провести отладку, для проверки правильности работы программы.

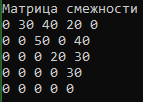


Рисунок 9 – Матрица смежности №1

На рисунке 9 изображена матрица смежности, по которой будет проводиться отладка. В первой сточке нет ребра в исток, поэтому будет выбрано первое попавшееся ребро, а именно ребро во вторую вершину, когда i равно 1.

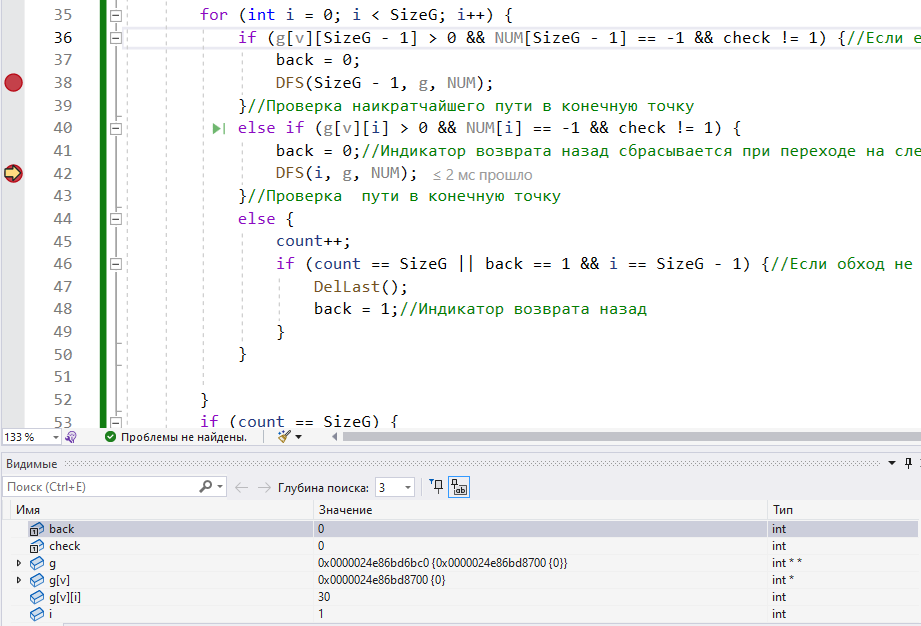


Рисунок 10 – Отладка

На рисунке 10 видно, что функция вызывается при значении i равном 1, что соответствует предполагаемому результату, дальше алгоритм переходит на 2 вершину, откуда есть путь в сток.

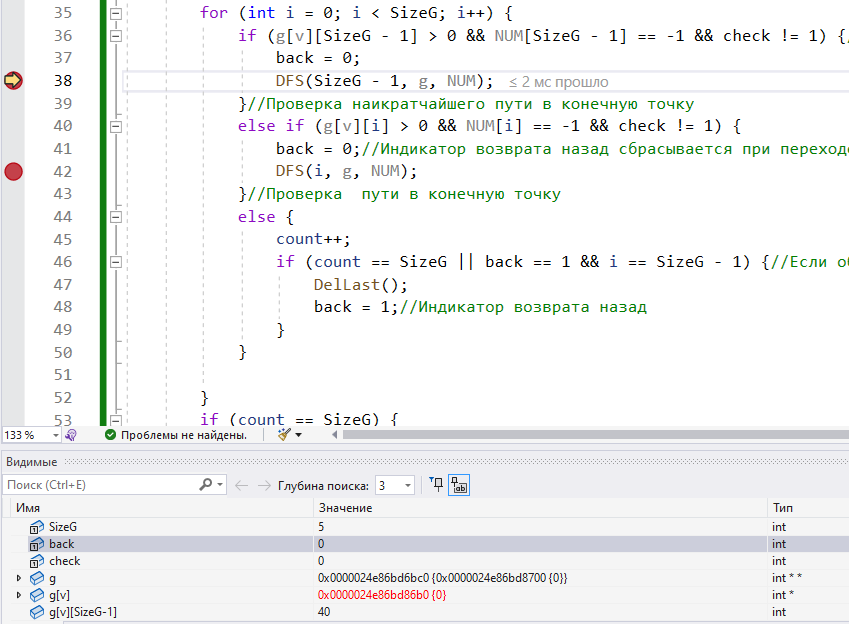


Рисунок 11 – Отладка

На рисунке 11 видно, что первое условие прошло, поэтому алгоритм перемещается в сток.

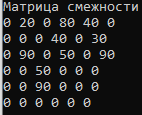


Рисунок 12 – Матрица смежности №2

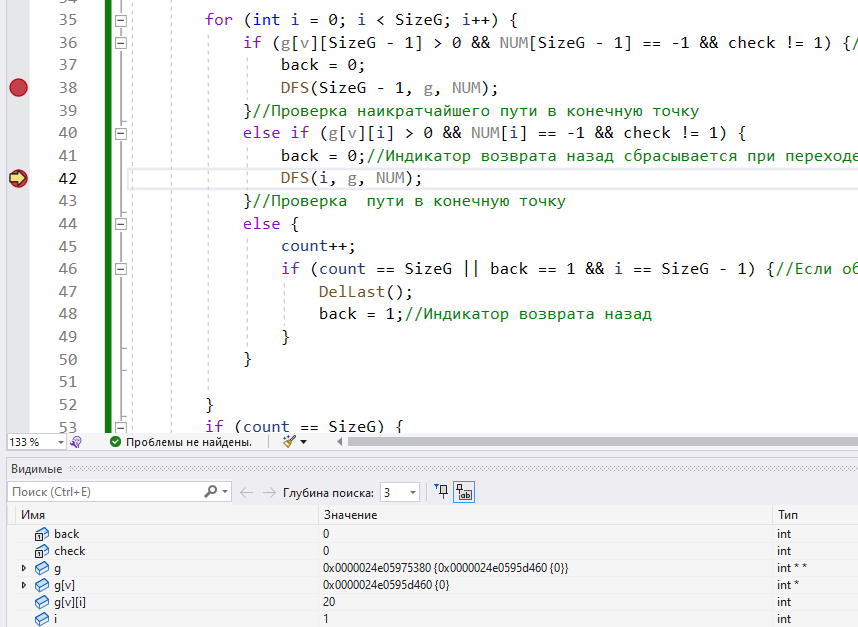


Рисунок 13 – Отладка

Выполним отладку на другом графе, сгенерированном случайным образом (Рисунок 12). На первой строчке нет пути в исток, поэтому он перешел на вторую вершину, i равно 1 (Рисунок 13), значит обход проходит правильно.

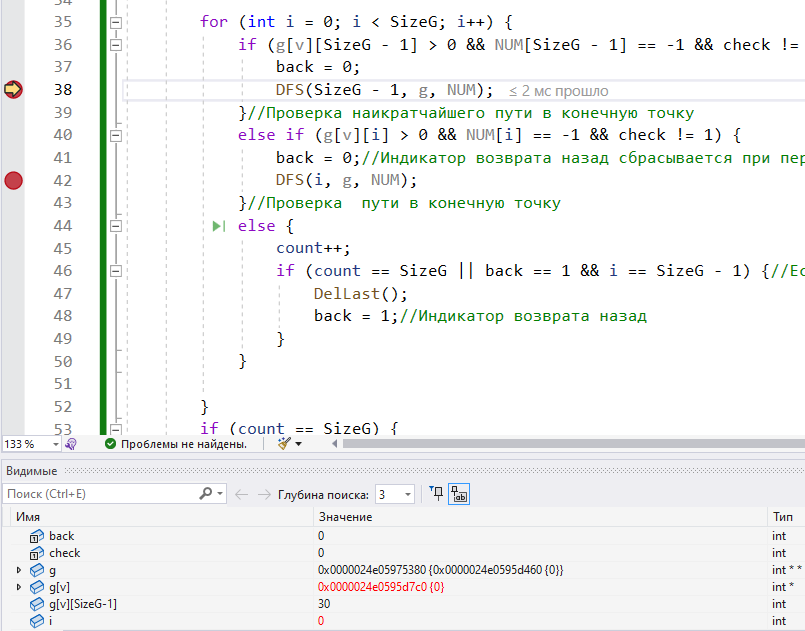


Рисунок 14 – Отладка

Из второй вершины можно перейти в сток, значит программа выбирает этот путь (Рисунок 14).

Отладка помогала в разработке программы, она применялась для исправления ошибок и совершенствования алгоритма.

# **6 Тестирование**

Тестирование проводилось в рабочем порядке, в процессе разработки,

после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при случайном заполнении графа, при 6 вершинах.

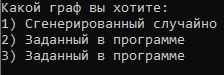


Рисунок 15 – Тестирование

1. Программа предлагает пользователю выбрать, с каким графом тот

хочет работать (Рисунок 15). Выбираем ввод вручную.

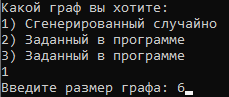


Рисунок 16 – Тестирование

1. У пользователя запрашивают размер графа (Рисунок 15). Вводим 6.

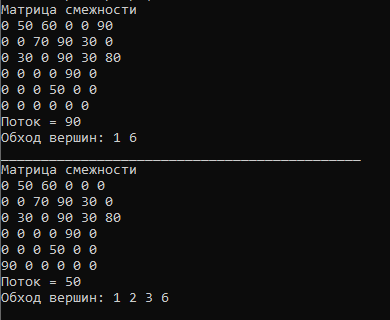


Рисунок 17 – Тестирование

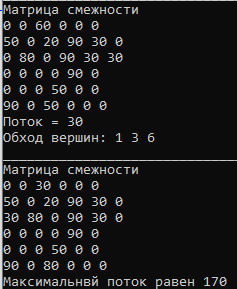


Рисунок 18 – Тестирование

1. На рисунках 17-18 видно, что программа выполнила 4 обхода, завершилась и вывела максимальный поток. Других путей из истока в сток нет, программа работает правильно.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Предоставление выбора генерации графа | Верно |
| Выбор генерации графа | Предоставление выбора размера графа | Верно |
| Выбор размера графа | Вывод матрицы смежностей, обход, нахождение потока, изменение матрицы смежности и повторение до конца | Верно |
| Отсутствие пути из истока в сток | Вывод максимального потока | Верно |

В результате тестирования было выявлено, что программа успешно выполняет функцию нахождения максимального потока.

# **7 Ручной растёт задачи**

Для ручного расчета используем граф, представленный в тестировании (Рисунок 17).

Начинается обход с первой вершины, так как есть ребро, соединяющее исток и сток, поток добрался до стока со значением 90. Вычитаем 90 из элемента матрицы смежности с параметрами (1;6) и прибавляем к (6;1).

Второй обход начинается в первой вершине и переходит во вторую, пропускная способность равно 50. После обход переходит в третью вершину, пропускная способность ребра больше, поток не изменяется. Из третей вершины можно перейти в сток, поток равен 50. Вычитаем 50 из элементов матрицы (1;2), (2;3), (3;6) и прибавляем 50 к (2;1), (3;2), (6;3).

Третий обход начинается в 1 вершине и переходит в 3, со значением потока равным 30, из 3 вершины можно добраться в 6. Вычитаем 30 из элементов матрицы (1;3), (3;6) и прибавляем 50 к (3;1), (6;3).

Больше путей из истока в сток нет, подсчитываем максимальный поток: 30+50+90=170. Результат ручных подсчетов сошелся с подсчетом программы. Программа работает правильно.

# **Заключение**

В ходе данной курсовой работы была исследована и реализована одна из задач теории графов - алгоритм Форда-Фалкерсона для нахождения максимального потока. Было проведено исследование теоретической основы этого алгоритма.

В процессе разработки программы, основанной на языках программирования C и C++, была продемонстрированы понимание алгоритма Форда-Фалкерсона и навыки программирования на языках Си и С++.

В результате выполнения этой курсовой работы были получены ценные знания в области алгоритмов теории графов и их реализации на практике. Работа над этой темой дала возможность углубиться в теоретические и практические аспекты алгоритма Форда-Фалкерсона и сформировать представление о его основных принципах и методах применения.

# **Список используемых источников**

1. Оре О. Графы и их применение. М., 2002. 171 с.
2. Campbell Parallel Programming with Microsoft® Visual C++. Москва: Гостехиздат, 2011. - 784 c.

# **Приложение А (Листинг программы)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

#include<time.h>

#include<stack>

#include <queue>

#include<Windows.h>

using namespace std;

queue<int> Q;//Очередь для хранения пройденный вершин

queue<int> Q2;

int SizeG, check, back = 0;

void DelLast() {//Удаляет послдний элемент в списке

int x = Q.size() - 1;

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

Q2.push(Q.front());

Q.pop();}

Q.pop();

while (!Q2.empty()) {

Q.push(Q2.front());

Q2.pop();}}

int FordFulkersonAlg (int v, int\*\* g, int\* NUM) {

NUM[v] = 1;

Q.push(v);

if (v != SizeG - 1) {//Проверка на достижение конечной вершины

int count = 0;

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

if (g[v][SizeG - 1] > 0 && NUM[SizeG - 1] == -1 && check != 1) {//Если есть путь в конечную вершину, то программа сначала перейдет в него

back = 0;

FordFulkersonAlg (SizeG - 1, g, NUM);

}//Проверка наикратчайшего пути в конечную точку

else if (g[v][i] > 0 && NUM[i] == -1 && check != 1) {

back = 0;//Индикатор возврата назад сбрасывается при переходе на следующую вершину

FordFulkersonAlg (i, g, NUM);

}//Проверка пути в конечную точку

else {

count++;

if (count == SizeG || back == 1 && i == SizeG - 1) {//Если обход не нашел вершину для перехода, он удаляет последний элемент из списка и возвращается назад

DelLast();

back = 1;//Индикатор возврата назад

}}}

if (count == SizeG) {

return 0;}}

else { //если достигнута конечная точка, то программа завершается, возвращая единицу, для продолжения поска потока

check = 1;

return 1;}}

void review(int\*\* G) {//Вывод матрицы

cout << "Матрица смежности\n";

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

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

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

cout << "\n";}}

int main() {

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

int\*\* G, \* DIST, vershina, vibor, max = 0, vivod, GO = 1, maxPotok = 0, c = 0, count;

cout << "Какой граф вы хотите:\n1) Сгенерированный случайно\n2) Заданный в программе\n3) Заданный в программе\n";

cin >> check;

switch (check) {

case(1)://случайная генерация массива

cout << "Введите размер графа: ";

cin >> SizeG;

break;

case(2)://заданная генерация с обычным обходом

SizeG = 5;

break;

case(3)://заданная генерация с возвратом на предыдущие вершины и очисткой очереди

SizeG = 5;

break;

default:

cout << "\n!ОШИБКА!";}

DIST = new int[SizeG];//массив пройденных вершин

G = new int\* [SizeG]; // создаём двумерный массив

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

G[i] = new int[SizeG];

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

G[i][j] = 0;}}

if (check == 3) { G[0][1] = 60, G[1][2] = 30, G[0][3] = 60, G[3][4] = 60; }

if (check == 2) { G[0][3] = 20, G[2][3] = 20, G[0][1] = 30, G[2][4] = 30, G[3][4] = 30, G[0][2] = 40, G[1][4] = 40, G[1][2] = 50; }

if (check == 1) {

for (int i = 0; i < SizeG - 1; i++) {

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

if ((rand() % 10) > 3 && i != j) {//заполнение случайным числами

G[i][j] = (rand() % 10) \* 10;}}}

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

count = 0;

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

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

count++;}}

if (count == SizeG) {

G[i][SizeG - 1] == (rand() % 10) \* 10;}}

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

count = 0;

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

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

count++;}}

if (count == SizeG) {

G[1][i] == (rand() % 10) \* 10;}}}

check = 0;

review(G);

while (GO) {

c++;

while (!Q.empty()) {

Q.pop();}

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

DIST[i] = -1;}

GO = FordFulkersonAlg (0, G, DIST);

check = 0;

if (GO != 0 && !Q.empty()) {

int Minimum = 0, operand1 = Q.front(), operand2;

Q2.push(operand1);

Q.pop();

while (!Q.empty()) {

operand2 = Q.front();

Q.pop();

if (G[operand1][operand2] < Minimum) {

Minimum = G[operand1][operand2];}

else if (Minimum == 0) {

Minimum = G[operand1][operand2];}

operand1 = operand2;

Q2.push(operand1);}

cout << "Поток = " << Minimum << "\n";

while (!Q2.empty()) {

Q.push(Q2.front());

Q2.pop();}

maxPotok += Minimum;

operand1 = Q.front();

cout << "Обход вершин: " << Q.front() + 1;

Q.pop();

while (!Q.empty()) {

operand2 = Q.front();

cout << " " << Q.front() + 1;

Q.pop();

G[operand1][operand2] -= Minimum;

G[operand2][operand1] += Minimum;

operand1 = operand2;}

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

review(G);}

if (c > SizeG+1) {

GO = 0;}}

cout << "Максимальнвй поток равен " << maxPotok << "\n";

system("pause");

return 0;}