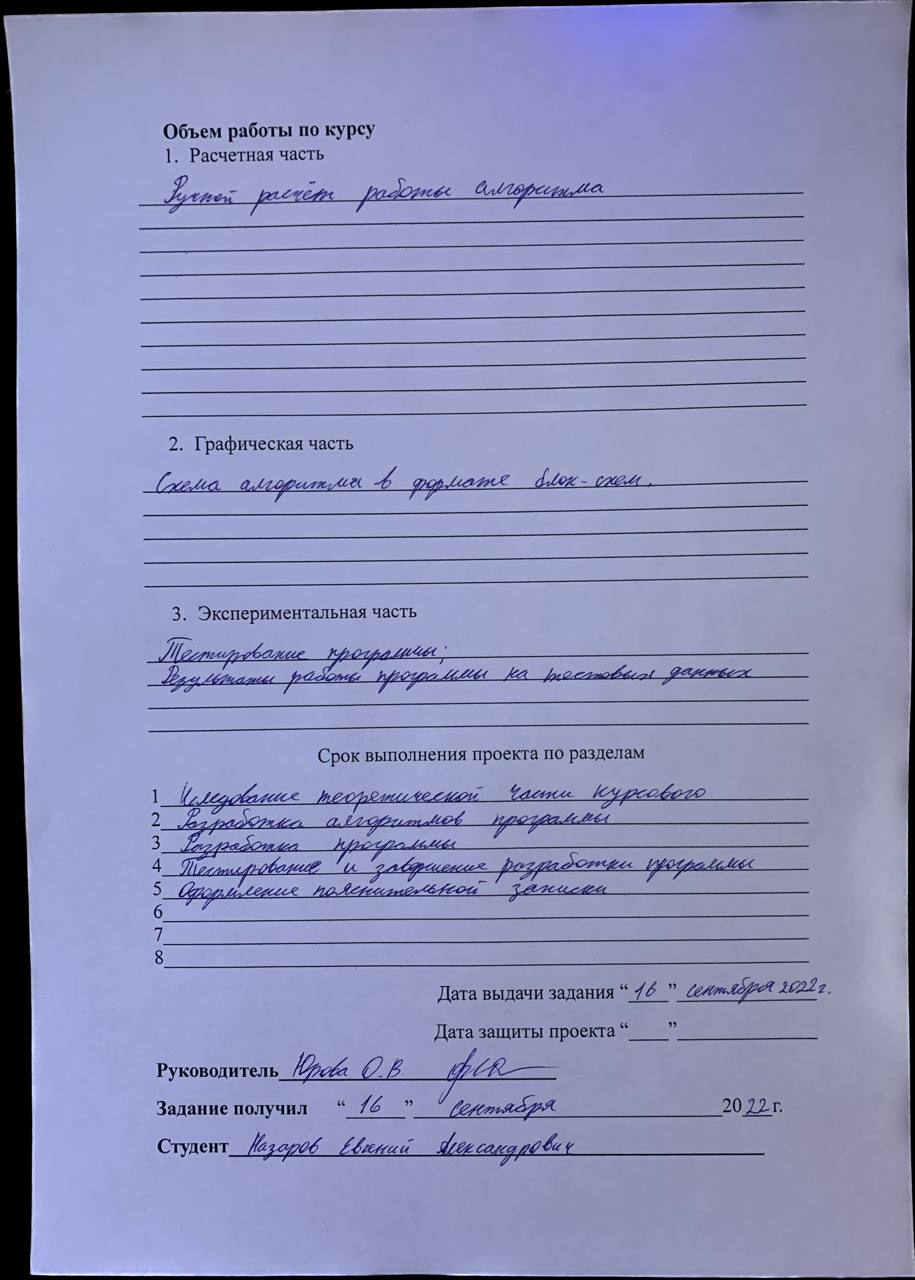
  
****

**Содержание**

[Реферат 5](#_Toc123109149)

[Введение 6](#_Toc123109151)

[1. Постановка задачи 8](#_Toc123109152)

[2. Теоретическая часть задания 9](#_Toc123109153)

[3. Описание алгоритма программы 12](#_Toc123109154)

[4. Описание программы 14](#_Toc123109155)

[5. Тестирование 19](#_Toc123109156)

[6. Ручной расчёт задачи 25](#_Toc123109157)

[Заключение 28](#_Toc123109159)

[Список литературы 29](#_Toc123109160)

[Листинг программы 30](#_Toc123109161)

# Реферат

# Отчет 35 стр, 20 рисунка.

НАИБОЛЬШЕЕ ПАРОСЧИТАНИЕ, ГРАФЫ, МАТРИЦА СМЕЖНОСТ.

Цель курсовой работы – осуществить программную реализацию алгоритма нахождения наибольшего паросочетания в заданном графе.

Программа должна работать так, чтобы пользователь вводил порядок графа. После выбора способа заполнения и обработки данных на экран должна выводиться матрица смежности, сгенерированная случайным самостоятельным образом или введённым пользователем. Проанализировав, данные новой матрицы программа должна находить наибольшее независимое множество, которое и будет являться наибольшим паросочетанием исходного графа.

# Введение

Начало теории графов как математической дисциплины было положено Эйлером в его знаменитом рассуждении о Кенигсбергских мостах. Однако эта статья Эйлера 1736 года была единственной в течение почти ста лет. Интерес к проблемам теории графов возродился около середины прошлого столетия и был сосредоточен главным образом в Англии. Имелось много причин для такого оживления изучения графов. Естественные науки оказали свое влияние на это благодаря исследованиям электрических цепей, моделей кристаллов и структур молекул. Развитие формальной логики привело к изучению бинарных отношений в форме графов. Большое число популярных головоломок подавалось формулировкам непосредственно в терминах графов, и это приводило к пониманию, что многие задачи такого рода содержат некоторое математическое ядро, важность которого выходит за рамки конкретного вопроса. Наиболее знаменитая среди этих задач–проблема четырех красок, впервые поставленная перед математиками Де Морганом около 1850 года. Никакая проблема не вызывала столь многочисленных и остроумных работ в области теории графов.

Настоящее столетие было свидетелем неуклонного развития теории графов, которая за последние десять – двадцать лет вступила в новый период интенсивных разработок. В этом процессе явно заметно влияние запросов новых областей: теории игр и программирования, теории передачи сообщений, электрических сетей и контактных цепей, а также проблем психологии и биологии.

Вследствие этого развития предмет теории графов является уже обширным, что все его основные направления невозможно изложить в одном томе. В настоящем первом томе предлагаемого двухтомного труда сделан акцепт на основные понятия и на результаты, вызывающие особый систематический интерес.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования – С++.

Целью данной курсовой работы является разработка программы на языке С++, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм нахождения наибольшего паросочетания.

# Постановка задачи

Исходный граф в программе должен задаваться матрицей смежности, причём при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, пользователь вводит количество вершин для генерации матрицы смежности, выбирает способ заполнения матрицы смежности, случайным способом или заполнить самому. После обработки этих данных на экран должна выводиться матрица смежности графа и наибольшее паросочитание. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Устройство ввода – клавиатура и мышь.

Задания выполняются в соответствии с вариантом №27.

# Теоретическая часть задания

Графом называется набор точек (эти точки называются вершинами), некоторые из которых объявляются смежными (или соседними). Считается, что смежные вершины соединены между собой ребрами (или дугами). Таким образом, ребро определяется парой вершин. Два ребра, у которых есть общая вершина, также называются смежными (или соседними).

В теории графов паросочетание или независимое множество рёбер в графе — это набор попарно несмежных рёбер.

Паросочетанием M называется набор попарно несмежных рёбер графа (иными словами, любой вершине графа должно быть инцидентно не более одного ребра из M).

Все вершины, у которых есть смежное ребро из паросочетания (т.е. которые имеют степень ровно один в подграфе, образованном M), назовём насыщенными этим паросочетанием.

Мощностью паросочетания назовём количество рёбер в нём. Наибольшим (максимальным) паросочетанием назовём паросочетание, мощность которого максимальна среди всех возможных паросочетаний в данном графе, а совершенным — где все вершины левой доли им насыщенны. Паросочетания можно искать в любых графах, и он работает за O(n3)

Понятие реберного графа для данного графа настолько естественно, что независимо было введено многими авторами. Конечно, каждый из них давал свое название: Ope назвал этот граф «смежностным графом», Сабидусси - «графом производной», Байнеке — «производным графом», Сешу и Рид - «реберно-вершинно-двойственным», Кастелейн — «покрывающим графом», Менон - «присоединенным» («сопряженным»). Были даны различные описания реберных графов. В этой главе вводится также тотальный граф, который изучался впервые Бехзадом , и поскольку (это очень удивительно!) он был обнаружен единожды, он не имеет других названий. Мы исследуем связь между реберными и тотальными графами, уделяя особое внимание эйлеровым и гамильтоновым графам.

Независимое множество вершин (известное также как внутренне устойчивое множество) есть множество вершин графа G, такое, что любые две вершины в нем не смежны (никакая пара вершин не соединена ребром). Внутренне устойчивое множество называется максимальным, если оно не является собственным подмножеством некоторого другого независимого множества. Наибольшее по мощности независимое множество называется наибольшим.

Свойства графа G, зависящие только от смежности рёбер, могут быть переведены в эквивалентные свойства графа L(G), зависящие только от смежности вершин. Например, паросочетание в G — это множество дуг, ни одна из которых не смежна другой, и соответствующее множество вершин в L(G), ни одна из которых не смежна другой, то есть независимое множество вершин.

* Рёберный граф связного графа связен. Если G связен, он содержит путь, соединяющий любые два его ребра, что переводится в путь графа L(G), содержащий любые две вершины графа L(G). Тем не менее, графу G, содержащему изолированные вершины, а посему несвязному, может соответствовать связный рёберный граф.
* Задача о максимальном независимом множестве для рёберного графа соответствует задаче нахождения максимального паросочетания в исходном графе.

Поскольку максимальное паросочетание может быть найдено за полиномиальное время, то и максимальное независимое множество рёберного графа может быть найдено за полиномиальное время вопреки трудности поиска такого множества для более общих семейств графов.

Независимые множества ребер графа G находятся во взаимно однозначном соответствии с независимыми множествами вершин реберного графа L(G)=(V1,X1), который для графа G=(V,X) определяется следующими двумя условиями:

1. V1 = X,

2. вершины х1 и х2 смежны в L(G) тогда и только тогда, когда ребра х1 и х2 смежны в G.

На рисунке 1 изображены два графа – G и L(G). Вершины графа G – темные кружки, вершины графа L(G) – светлые кружки. Ребра графа G – тонкие линии, ребра графа L(G) – жирные линии.

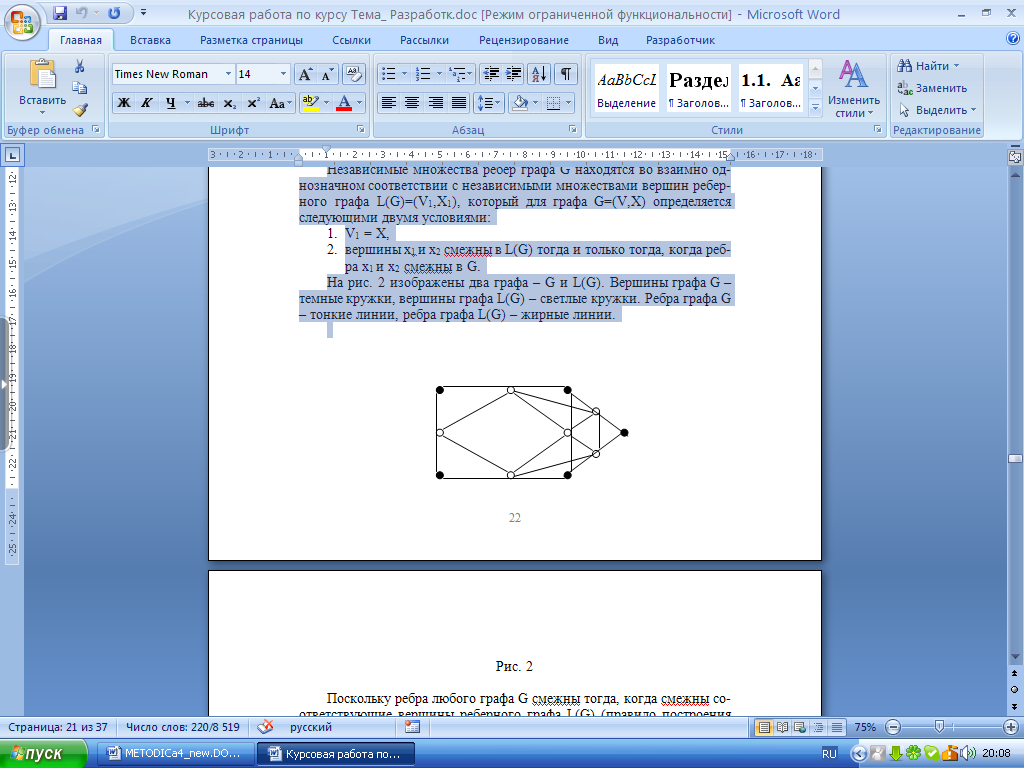
******

Рисунок 1

Поскольку ребра любого графа G смежны тогда, когда смежны соответствующие вершины реберного графа L(G) (правило построения реберного графа смотри ниже), то χ ‘(G) можно определить как хроматическое число графа L(G), т.е. χ ‘(G)=χ (L(G)).

К сожалению, не существует оптимального алгоритма. Задачу можно решить либо полным перебором, либо с приближенным значением.

# Описание алгоритма программы

После ввода порядка графа пользователем формируется матрица смежности, заполненная случайными значениями. Далее вызывается функция findAllMatching для нахождения всех паросочетаний в графе.

Следом функция findAllMatchings, возвращает все паросочетания относительно переданного ребра. Далее перебираем все ребра графа, а также попарно проверяем текущее ребро со всеми остальными. После В найденном векторе ищем максимальное паросочетание. Функция findMaxMatchingFromVector, находит наибольшее паросочетание переданных ребер. Запускаем цикл для проверки рёбер на смежность. Если хоть одно ребро из вектора является не смежным относительно n-ого графа. Рекурсивно проверяем не смежные относительно n-ого ребра на смежность между друг другом. Если количество не смежных ребер оказалось больше чем предыдущее, то записываем значение newMaxMatchings переменной maxMatchings. Добавляем перебираемое ребро в конец списка. Если новый вектор больше старого по размеру, то присваиваем значение newMaxMatching переменной maxMatching. В конце выводим матрицу и вектор рёбер на экран.

Ниже представлен псевдокод функции типа Edge динамической структуры данных Vector “findAllMatchings":

1. Создать динамическую функцию mathcings.
2. Создать контейнер edges
3. Присвоение значения графа graph.edge
4. Цикл для graphEdge=первое значение edges пока graphEdge не равно концу списка edges.
5. Вектору mathcings присвоить значение findAllMatchings.
6. Вектору newMaxMatching присвоить значение findMaxMatchingFromVector.
7. В конец списка newMaxMatching добавляем graphEdge
8. Добавление в конец списка graphEdge newMaxMatching
9. Если длина newMaxMatching больше чем длина maxMatching
10. То присваиваем maxMatching значение newMaxMatching
11. Цикл для edge=первое значение maxMatching пока edge не ровно концу списка maxMatching.
12. Вывод вершин ребра.

# Описание программы

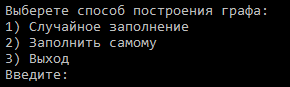
Для написания данной программы использован язык программирования С++. Язык программирования С++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

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

Работа программы начинается с ввода количества вершин.



Потом перед пользователем стоит выбор в главном меню.



Если пользователь выбрал заполнения матрицы случайным образом ему придётся ввести вероятности присутствия ребра между вершинами.

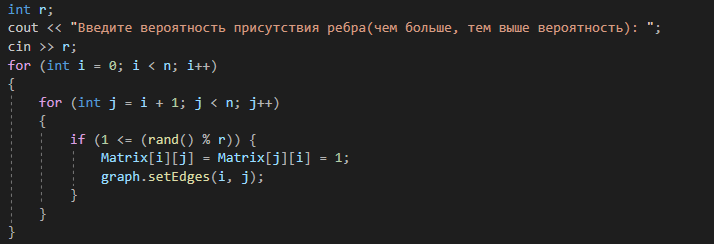


Рисунок 2

Если же пользователь выбрал заполнения матрицы самому ему придётся вводить поочередную связь между вершинами, пока он самостоятельно не выйдет из этого из цикла.

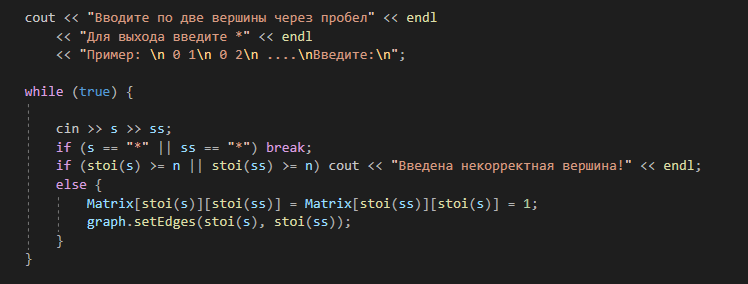


Рисунок 3

Структура Node хранит имя вершины.



Рисунок 4

Структура Edge хранит информацию о дуге, храня в себе 2 указателя на вершины в этой дуге.

Перегрузка оператора “==” служит для проверки равенства двух рёбер.

Перегрузка оператора “<” служит для добавления в множество.

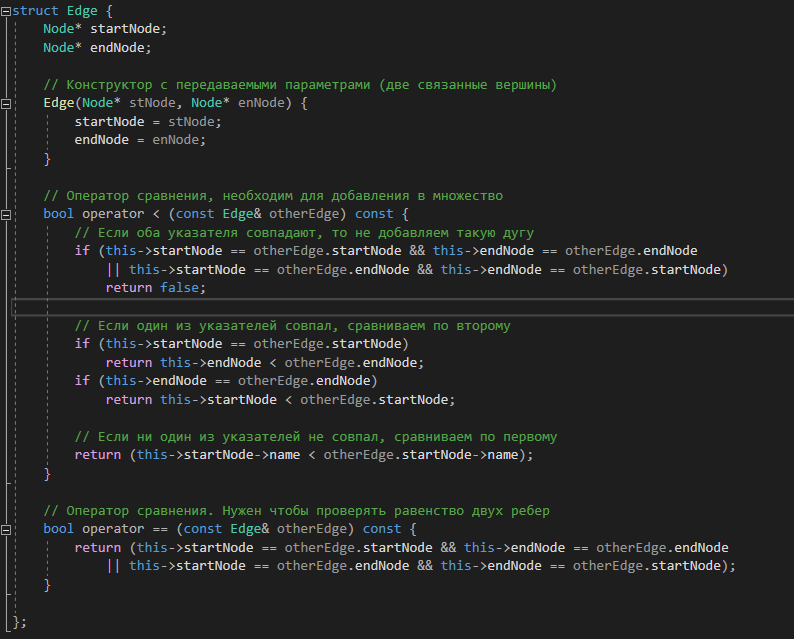


Рисунок 5

Структура Graph служит для хранения всей информации о графе.

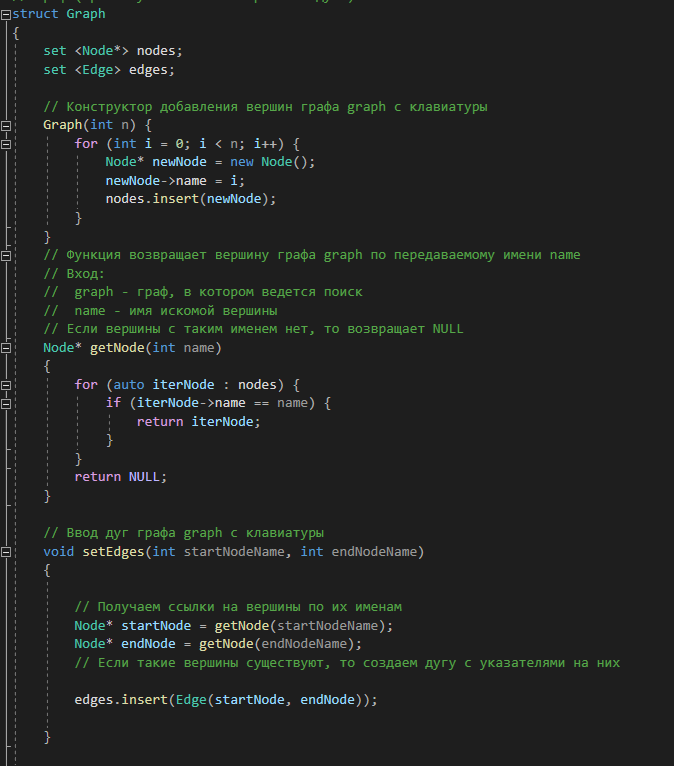


Рисунок 6

Конструктор принимает в себя количество вершин графа ведённое пользователем и выделяет нужное количество динамической памяти, для корректной работы

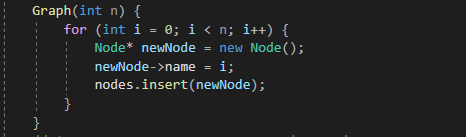


Рисунок 7

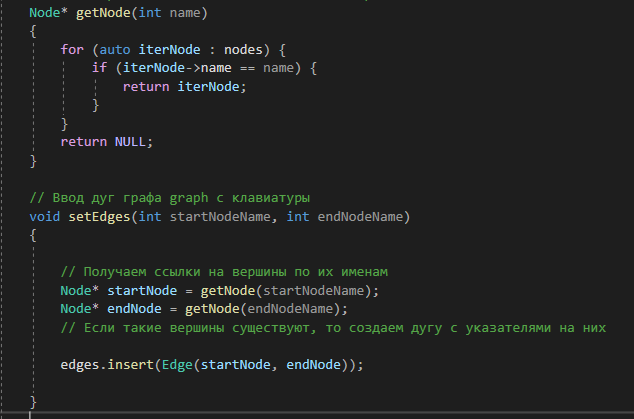
Функция setEdge служит для добавления ребра в граф. Для этого ей понадобится помощь функции getNode, которая в свою же очередь возвращает вершину графа по переданному имени.

Рисунок 8

# Тестирование

Среда разработки Microsoft Visual Studio 2019 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 было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин и вывода результата.

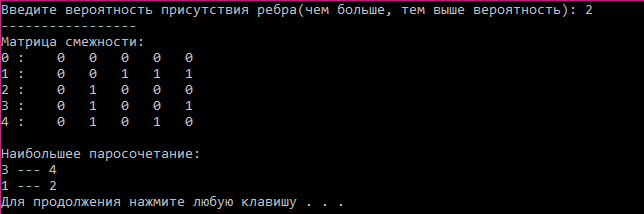


Рисунок 9 – тестирование при случайном заполнении графа при 5 вершинах и вероятности присутствия ребра равному 0.5

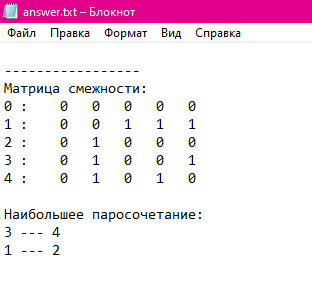


Рисунок 10 – вывод тестирования сверху в файл

Изменим количество вершин и вероятность присутствия вершин.

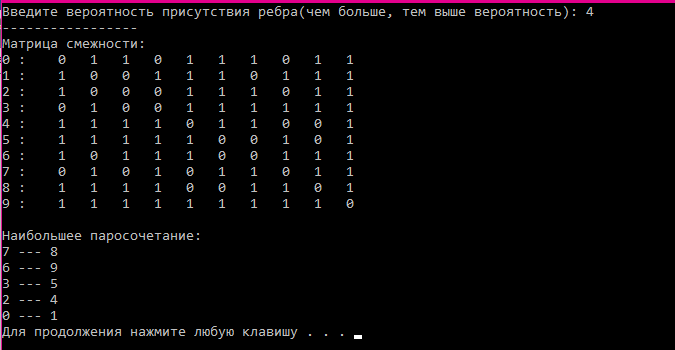


Рисунок 11 – тестирование при случайном заполнении графа при 9 вершинах и вероятности присутствия ребра равному 0.25

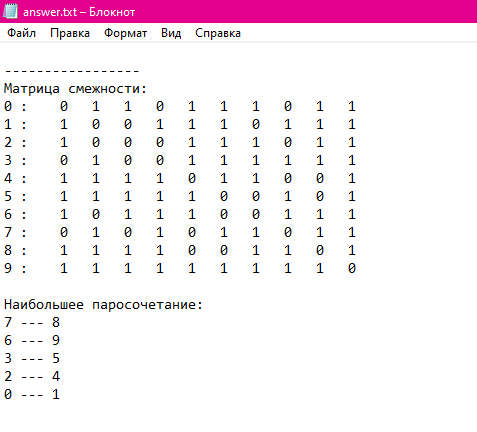


Рисунок 12 – вывод тестирования сверху в файл

Теперь введем связь вершин самостоятельно.

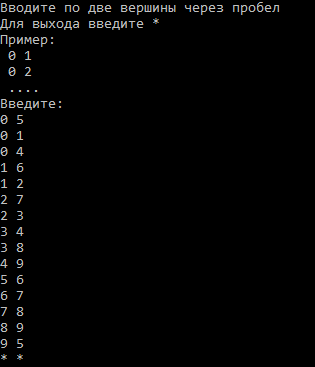


Рисунок 13 – ввод связи между вершинами

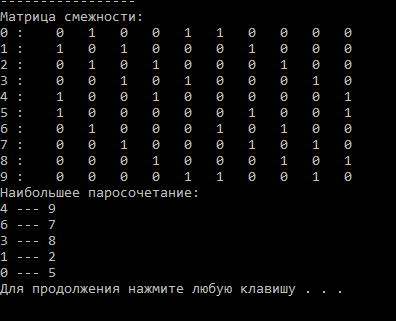


Рисунок 14 – результат тестирования введенных данный для 10 вершин

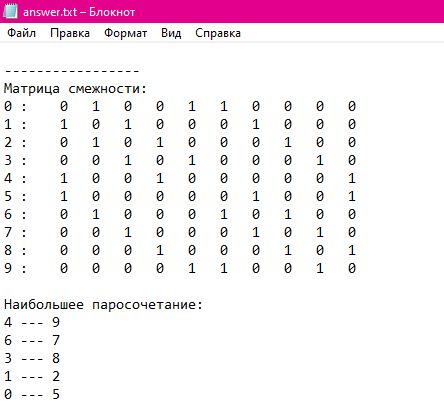


Рисунок 15 - вывод тестирования сверху в файл

Изменим количество вершин графа и протестируем по новой.

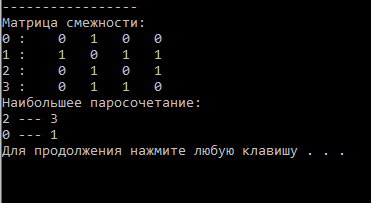


Рисунок 16 – ввод связи вершин и вывод результата для 4 вершин

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод сообщения о вводе кол-ва вершин | Верно |
| Выбор генерации матрицы | Вывод меню программы и выбор пользователем нужного пункта | Верно |
| Заполнение матрицы случайными числами | Ввод вероятности и вывода матрица смежности | Верно |
| Самостоятельное и случайное заполнение матрицы смежности, корректный вывод результата | Вывод матрицы смежности с n вершинами. Вывод результата в виде раскраски графа и цветов каждой вершины соответствуя алгоритму. | Верно |

Таблица 1

В результате тестирования было выявлено, что программа успешно выдает результат.

# Ручной расчёт задачи

Мы имеем матрицу смежности вершин 4х4:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |

Представим матрицу в графическом виде для большей наглядности. Смотреть на рисунке 17.

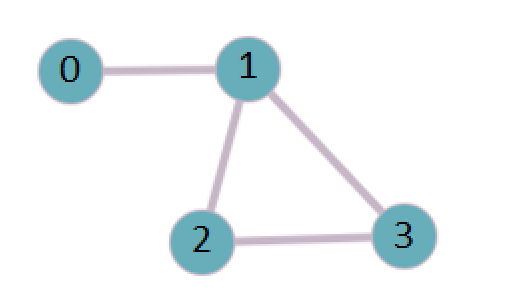


Рисунок 17

С помощью перебора ищем наибольшее паросочитание графа.

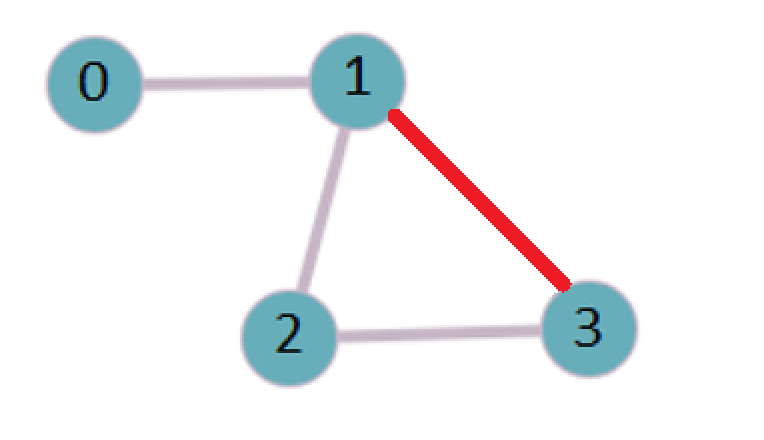


Рисунок 18

В этом случае мы можем найти только одно паросочитание [1-3].

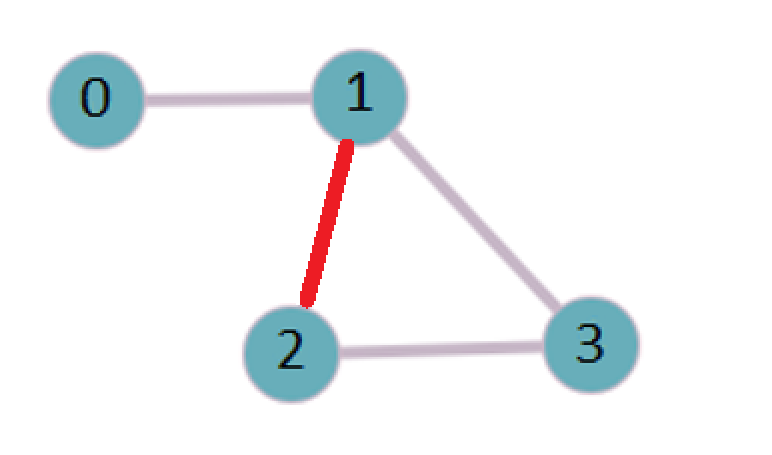


Рисунок 19

В этом случае мы можем найти только одно паросочитание [1-2].

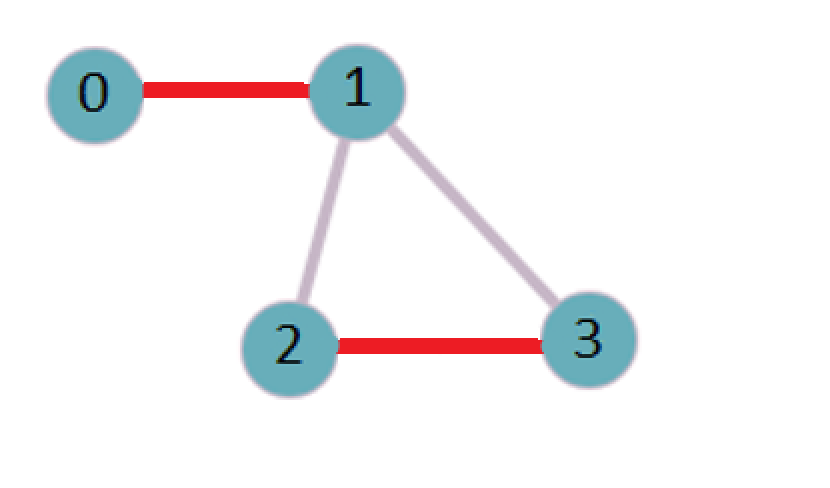


Рисунок 20

В этом случае мы можем найти два паросочитание [0-1 2-3].

Искомое наибольшее паросочетание:

[0-1 2-3]

# Результаты работы программы с аналогичными данными представлен на рисунке 16. Таким образом, можно сделать вывод, что программа работает верно.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска максимального паросочетания в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей, основанных на теории неопределенных графов. Приобретены навыки по осуществлению алгоритма нахождения максимально паросочетания в графе. Углублены знания языка программирования C++.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978
2. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006
3. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.

# Листинг программы

#include <iostream>

#include <set>

#include <string>

#include <fstream>

#include <vector>

#include <iomanip>

using namespace std;

ofstream fout("answer.txt");

// Вершина (хранит только свое имя)

struct Node {

int name;

};

// Дуга (хранит указатели на исходную и конечную вершины)

struct Edge {

Node\* startNode;

Node\* endNode;

// Конструктор с передаваемыми параметрами (две связанные вершины)

Edge(Node\* stNode, Node\* enNode) {

startNode = stNode;

endNode = enNode;

}

// Оператор сравнения, необходим для добавления в множество

bool operator < (const Edge& otherEdge) const {

// Если оба указателя совпадают, то не добавляем такую дугу

if (this->startNode == otherEdge.startNode && this->endNode == otherEdge.endNode

|| this->startNode == otherEdge.endNode && this->endNode == otherEdge.startNode)

return false;

// Если один из указателей совпал, сравниваем по второму

if (this->startNode == otherEdge.startNode)

return this->endNode < otherEdge.endNode;

if (this->endNode == otherEdge.endNode)

return this->startNode < otherEdge.startNode;

// Если ни один из указателей не совпал, сравниваем по первому

return (this->startNode->name < otherEdge.startNode->name);

}

// Оператор сравнения. Нужен чтобы проверять равенство двух ребер

bool operator == (const Edge& otherEdge) const {

return (this->startNode == otherEdge.startNode && this->endNode == otherEdge.endNode

|| this->startNode == otherEdge.endNode && this->endNode == otherEdge.startNode);

}

};

// Граф (хранит указатели на вершины и дуги)

struct Graph

{

set <Node\*> nodes;

set <Edge> edges;

// Конструктор добавления вершин графа graph с клавиатуры

Graph(int n) {

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

Node\* newNode = new Node();

newNode->name = i;

nodes.insert(newNode);

}

}

// Функция возвращает вершину графа graph по передаваемому имени name

Node\* getNode(int name)

{

for (auto iterNode : nodes) { // graph - граф, в котором ведется поиск

if (iterNode->name == name) { // name - имя искомой вершины

return iterNode;

}

}

return NULL; // Если вершины с таким именем нет, то возвращает NULL

}

// Ввод дуг графа graph с клавиатуры

void setEdges(int startNodeName, int endNodeName)

{

// Получаем ссылки на вершины по их именам

Node\* startNode = getNode(startNodeName);

Node\* endNode = getNode(endNodeName);

// Если такие вершины существуют, то создаем дугу с указателями на них

edges.insert(Edge(startNode, endNode));

}

};

// Являются ли хоть одна пара ребер смежными

bool hasMatchingPoints(vector<Edge> edges)

{

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

{

for (int j = i + 1; j < edges.size(); j++)

{

if (edges[i].startNode == edges[j].startNode ||

edges[i].startNode == edges[j].endNode ||

edges[i].endNode == edges[j].startNode ||

edges[i].endNode == edges[j].endNode)

return true;

}

}

return false;

}

// Функция возвращает все паросочетания относительно переданного ребра

vector<Edge> findAllMatchings(Graph graph, Edge edge)

{

vector<Edge> mathcings;

set <Edge> edges = graph.edges; // присвоение графа

// Перебираем все ребра графа

for (Edge graphEdge : edges)

{

if (!(graphEdge == edge))

{

// Попарно проверяем текущее ребро со всеми остальными

vector<Edge> checkEdges = { graphEdge, edge };

if (!hasMatchingPoints(checkEdges))

mathcings.push\_back(graphEdge);

}

}

return mathcings;

}

// Функция находит наибольшее паросочетание переданных ребер

vector<Edge> findMaxMatchingFromVector(vector<Edge> edges)

{

// Если все ребра не смежные

if (!hasMatchingPoints(edges))

return edges;

// Хранилище наибольшего паросочетания

vector<Edge> maxMatchings;

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

{

// Хранилище ребер, не смежных i-тому

vector<Edge> newEdges;

for (int j = i + 1; j < edges.size(); j++)

{

vector<Edge> checkEdges = { edges[i], edges[j] };

// Проверяем i-тое и j-тое ребра на смежность

if (!hasMatchingPoints(checkEdges))

newEdges.push\_back(edges[j]);

}

// Если хоть одно ребро из вектора является не смежным относительно i-того

if (newEdges.size() > 0)

{

// Рекурсивно проверяем не смежные относительно i-того ребра ребра на смежность между друг другом

vector<Edge> newMaxMatchings = findMaxMatchingFromVector(newEdges);

newMaxMatchings.push\_back(edges[i]);

// Если количество не смежных ребер оказалось больше чем предыдущее

if (newMaxMatchings.size() > maxMatchings.size())

maxMatchings = newMaxMatchings;

}

else

{

// Если до этого вектор несмежных ребер был пуст

if (1 > maxMatchings.size())

maxMatchings = { edges[i] };

}

}

return maxMatchings;

}

// Главная функция - нахождения наибольшего паросочетания в графе

void findMaxMatchingVector(Graph graph)

{

vector<Edge> maxMatching; // ответ хранится тут

set <Edge> edges = graph.edges;

// Перебираем все ребра

for (Edge graphEdge : edges)

{

// Находим все паросочетания отосительно перебираемого

vector<Edge> mathcings = findAllMatchings(graph, graphEdge);

// В найденном векторе ищем максимальное паросочетание

vector<Edge> newMaxMatching = findMaxMatchingFromVector(mathcings);

// Добавляем перебираемое ребро

newMaxMatching.push\_back(graphEdge);

// Если новый вектор больше старого по размеру

if (newMaxMatching.size() > maxMatching.size())

maxMatching = newMaxMatching;

}

// Вывод вектора ребер на экран (В векторе хранится наибольшее паросочетание)

for (Edge edge : maxMatching) {

cout << edge.startNode->name << " --- " << edge.endNode->name << endl;

fout << edge.startNode->name << " --- " << edge.endNode->name << endl;

}

}

// функция вывода матрицы

void toString(int\*\* Matrix, int n) {

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

{

cout << i << " : ";

fout << i << " : ";

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

cout << setw(4) << Matrix[i][j];

fout << setw(4) << Matrix[i][j];

}

cout << endl;

fout << endl;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int n;

string s, ss, menu;

while (true) {

cout << "Введите кол-во вершин графа: "; cin >> n;

Graph graph(n);

int\*\* Matrix = new int\* [n]; ;

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

Matrix[i] = new int[n];

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

{

Matrix[i][j] = 0;

}

}

cout << "\n Выберете способ построения графа: \n 1) Случайное заполнение \n"

<< " 2) Заполнить самому \n 3) Выход\n Введите: ";

cin >> menu;

switch (atoi(menu.c\_str()))

{

case 1:

system("cls");

int r;

cout << "Введите вероятность присутствия ребра(чем больше, тем выше вероятность): ";

cin >> r;

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

{

for (int j = i + 1; j < n; j++)

{

if (1 <= (rand() % r)) {

Matrix[i][j] = Matrix[j][i] = 1;

graph.setEdges(i, j);

}

}

}

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << endl << "-----------------" << endl << "Матрица смежности: \n";

toString(Matrix, n);

cout << endl << "Наибольшее паросочетание:" << endl;

fout << endl << "Наибольшее паросочетание:" << endl;

findMaxMatchingVector(graph);

system("pause");

break;

case 2:

system("cls");

cout << "Вводите по две вершины через пробел" << endl

<< "Для выхода введите \*" << endl

<< "Пример: \n 0 1\n 0 2\n ....\nВведите:\n";

while (true) {

cin >> s >> ss;

if (s == "\*" || ss == "\*") break;

if (stoi(s) >= n || stoi(ss) >= n) cout << "Введена некорректная вершина!" << endl;

else {

Matrix[stoi(s)][stoi(ss)] = Matrix[stoi(ss)][stoi(s)] = 1;

graph.setEdges(stoi(s), stoi(ss));

}

}

cout << "-----------------" << endl << "Матрица смежности: \n";

fout << endl << "-----------------" << endl << "Матрица смежности: \n";

toString(Matrix, n);

cout << "Наибольшее паросочетание:" << endl;

fout << endl << "Наибольшее паросочетание:" << endl;

findMaxMatchingVector(graph);

system("pause");

break;

case 3:

exit(0);

break;

}

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

delete[] Matrix[i];

delete[] Matrix;

system("cls");

}

fout.close();

return 0;

}