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 «Алгоритм поиска множества независимых

вершин в графе»»

Выполнил:

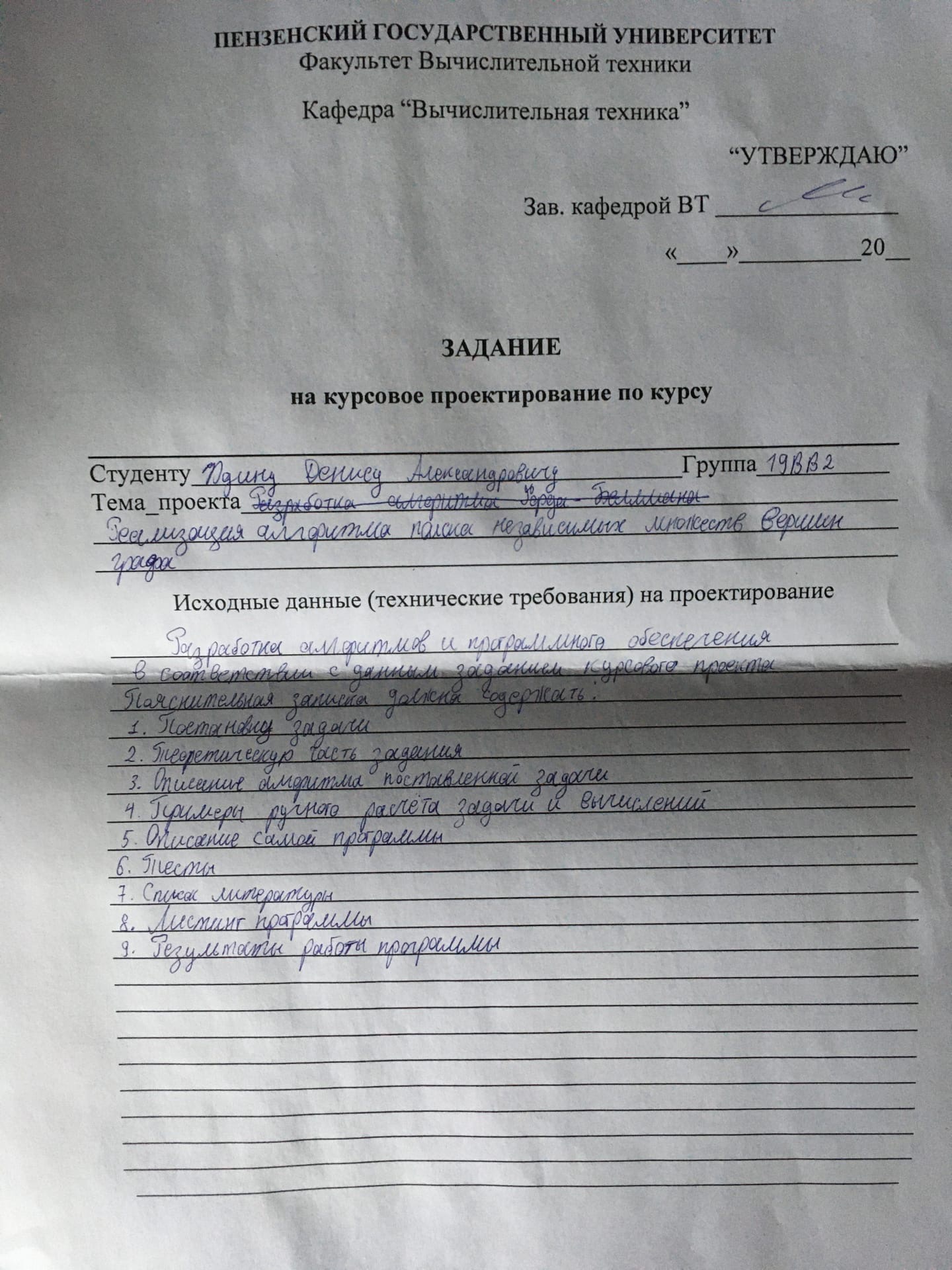
Юдин Д. А.

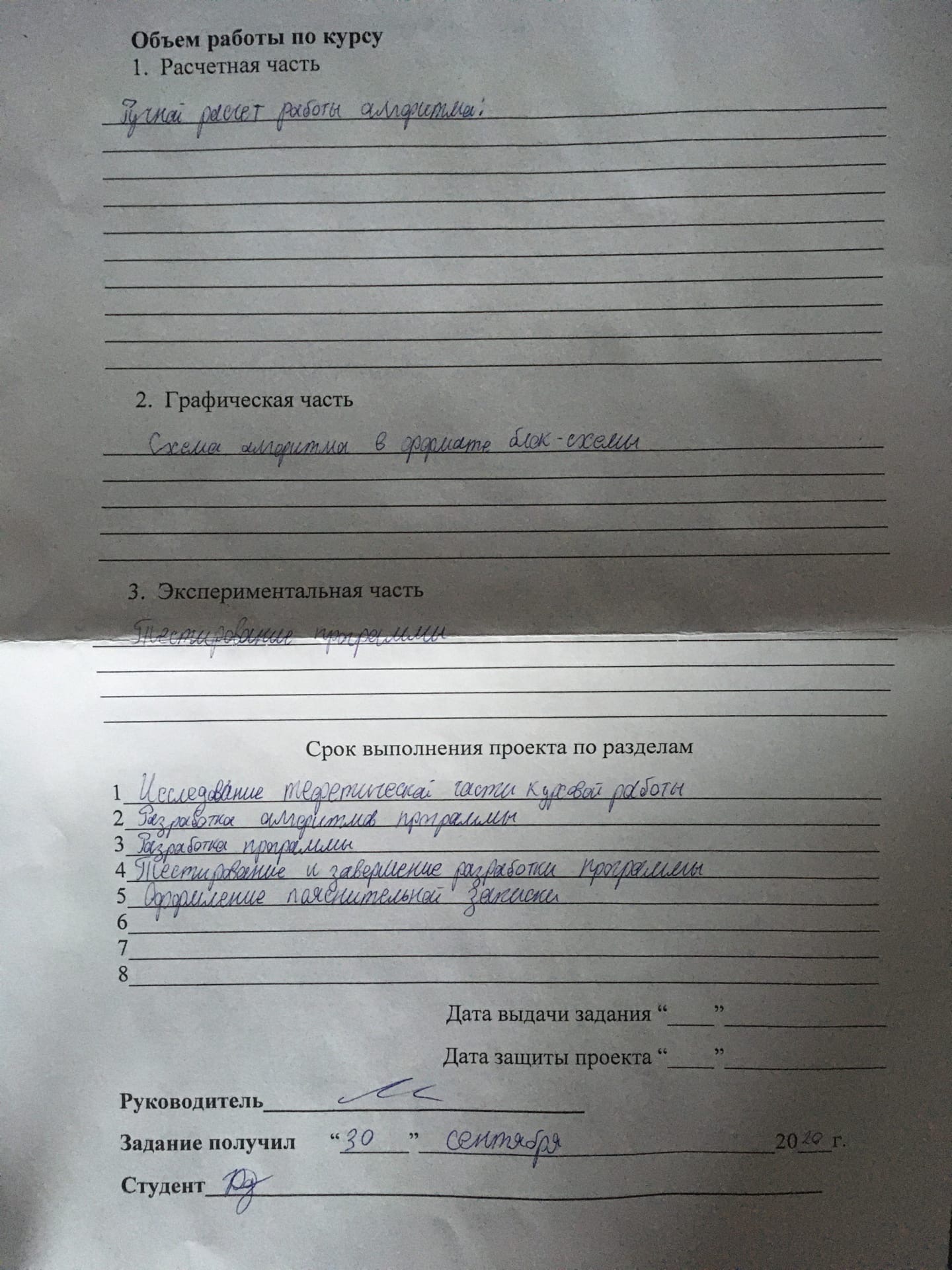
Приняли:

Митрохин М. А.

Юрова О.В.

Пенза 2020





**Содержание**

[Реферат 5](#_Toc59630238)

[Введение 6](#_Toc59630239)

[1. Постановка задачи 6](#_Toc59630240)

[2. Теоретическая часть задания 8](#_Toc59630241)

[3. Описание алгоритма программы 10](#_Toc59630242)

[5. Описание программы 17](#_Toc59630243)

[5.1 Общее описание 17](#_Toc59630244)

[5.2 Функции программы 17](#_Toc59630245)

[6. Тестирование 22](#_Toc59630246)

[7. Ручной расчет задачи 26](#_Toc59630247)

[Заключение 28](#_Toc59630248)

[Список литературы 29](#_Toc59630249)

[Приложение А.Листинг программы. 30](#_Toc59630250)

# Реферат

Отчет 33 стр., 20 рисунков

ГРАФ, АЛГОРИТМ ПОИСКА НЕЗАВИСИМЫХ МНОЖЕСТВ ВЕРШИН ГРАФА

Цель исследования – разработка программы, способной находить независимые множества вершин графа.

В работе рассмотрен граф, который представлен при помощи матрицы смежности. Разработан алгоритм поиска с возвратом для выполнения цели исследования.

# Введение

В данной курсовой работе необходимо составить программу, которая будет генерировать независимые множества заданного графа с помощью алгоритма с возвратом. Множество вершин графа называется **независимым**, если никакие две вершины этого множества не соединены ребром. Другими словами, индуцированный этим множеством подграф состоит из изолированных вершин. Алгоритм может быть полезен как для поиска множеств всех независимых вершин в графе, так и для поиска возможных клик в графе, если немного изменить условие его работы. Вычислительная сложность алгоритма Линейна относительно количества клик в графе. Исследования в 2006 году показали, что в худшем случае алгоритм работает за *O*(3*n*/3), где n — количество вершин в графе.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2019, язык программирования – С++. В ходе выполнения данной курсовой работы были приобретены навыки работы с формами и их элементами в среде Microsoft Visual Studio 2019, навыки работы с проектами и многомодульными программами.

# Постановка задачи

Задание для курсового проектирования заключается в создании программного продукта, позволяющего наглядно представить все наибольшие независимые множества вершин графа с помощью алгоритма с возвратом (англ.: backtracking). Программа должна будет реализовывать поиск наибольших независимых множеств графа, сгенерированных случайным образом.

Для удовлетворения требований задачи был выбрал язык C++. C++  —[компилируемый](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), [статически типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), так и [низкоуровневых языков](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). В сравнении с его предшественником — языком [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), — наибольшее внимание уделено поддержке [объектно-ориентированного](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [обобщённого программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), разнообразных прикладных программ, [драйверов](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80) устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это [GCC](https://ru.wikipedia.org/wiki/GNU_Compiler_Collection), [Visual C++](https://ru.wikipedia.org/wiki/Visual_C%2B%2B), [Intel C++ Compiler](https://ru.wikipedia.org/wiki/Intel_C%2B%2B_Compiler), [Embarcadero (Borland) C++ Builder](https://ru.wikipedia.org/wiki/Embarcadero_C%2B%2B_Builder) и другие.

Синтаксис C++ унаследован от языка [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как [компиляторами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

Исходный граф в программе должен задаваться матрицей смежности. Пользователь имеет возможность или вручную ввести ее размер, или сгенерировать автоматически. Устройство ввода – клавиатура и мышь.

# Теоретическая часть задания

Граф G (рисунок 1) задается множеством вершин X1, X2, ..., Xn. и множеством ребер, соединяющих между собой определенные вершины.

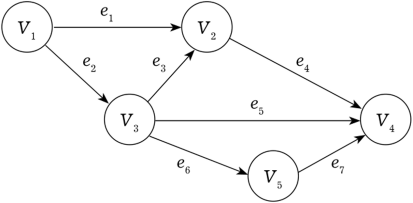


Рисунок 1 – пример графа G

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

Множество вершин U графа G = (V,X) называется независимым (внутренне устойчивым), если никакие две вершины из этого множества не смежны. Другими словами, если U ⊆V и U независимо в графе G, то порожденный подграф G(U) является пустым. Очевидно, что если при этом U\*⊆ U, то U\* - также независимое множество.

Очевидный алгоритм, который можно применить для нахождения независимых множеств вершин, это «полный перебор всех возможностей»: генерируем все возможные подмножества вершин заданного графа или орграфа и проверяем, является ли оно независимым.

Опишем теперь общий метод, позволяющий значительно сократить число шагов в алгоритмах полного перебора всех возможностей. Чтобы применить этот метод, представим искомое решение в виде последовательности <x1,…x n>. Основная идея метода состоит в том, что решение строится последовательно, начиная с пустой последовательности ε (длины 0). Вообще, имея данное частичное решение <x1,…x k>, мы стараемся найти такое допустимое значение x k+1, относительно которого мы не можем сразу сказать, что <x1,…x k, x k+1> можно расширить до некоторого решения (либо <x1,…x k, x k+1> уже является решением).

Если такое предполагаемое, но еще не использованное значение x k+1 существует, то мы добавляем его к нашему частичному решению и продолжаем процесс для последовательности <x1,…x k, x k+1>. Если его не существует, то мы возвращаемся к нашему частичному решению <x1,…x k-1> и продолжаем процесс, отыскивая новое, еще не использованное допустимое значение xk – отсюда название «алгоритм с возвратом» (англ.: backtrack).

# Описание алгоритма программы

Для программной реализации алгоритма с возвратом нам необходимо создать матрицу смежности, отображающую связь между вершинами в графе G = (V, E). Программа позволяет задать размер матрицы вручную или генерирует ее размер автоматически. В этом случае используется функция *rand*, которая позволяет случайно генерировать элемент для последующей записи. На ее основе генерируется случайное число, размер матрицы смежности определяется как остаток от деления этого числа на 9 (рисунок 2).

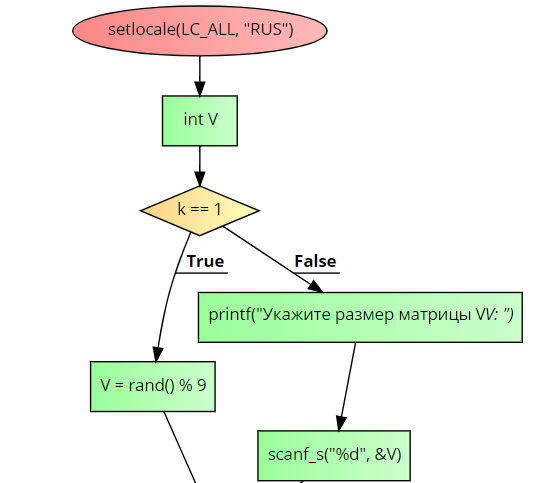


Рисунок 1 - блок-схема определения размера матрицы смежности

Для того, чтобы осуществлять проход по вершинам в ходе работы основного алгоритма мы создаем вектор vertices, куда заносим вершины графа (рисунок 3). Вектор в C++ — это замена стандартному динамическому массиву, память для которого выделяется вручную, с помощью оператора new.

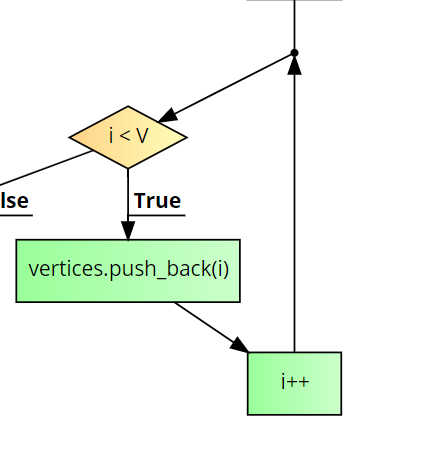


Рисунок 2 - блок-схема внесения вершин графа в vector vertices

Затем мы генерируем матрицу смежности для невзвешенного неориентированного графа. Единица в матрице смежности отображает, что между вершинами есть связь, ноль – отсутствие связи. Генерация матрицы происходит с помощью той же функции rand, но уже берется остаток по делению на 2 (рисунок 5). Для того, чтобы в графе не было петель, предусмотрено обнуление главной диагонали.

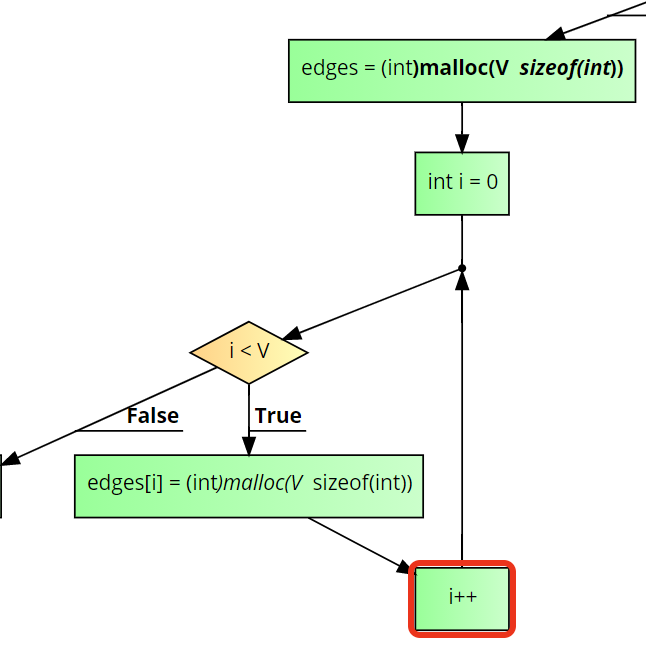


Рисунок 3 - выделения памяти для создания матрицы смежности

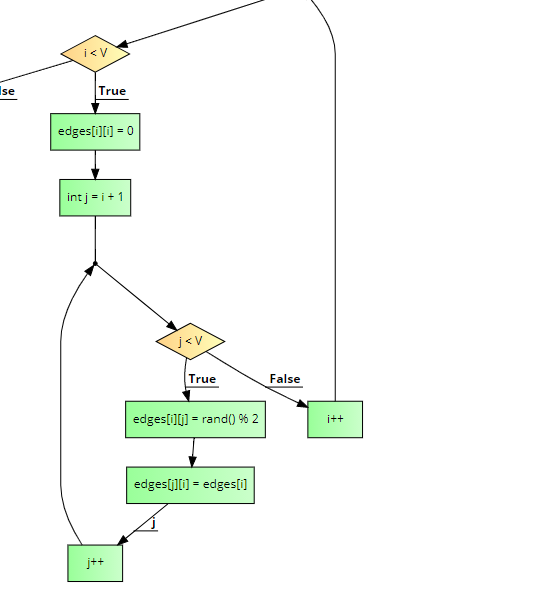


Рисунок 4 – Заполнение матрицы смежности

После того, как мы создали матрицу смежности, выводим ее на экран (рисунок 6).

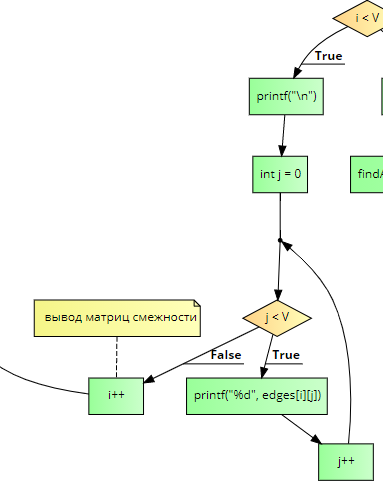


Рисунок 5 – Вывод матрицы смежности

После того, как мы вывели матрицу смежности на экран, создаем переменную tempSolutionSet типа set, в которой мы будем хранить все несмежные вершины при каждом рекурсивном проходе по алгоритму при просмотре новой вершины. Set — это контейнер, который автоматически сортирует добавляемые элементы в порядке возрастания. Затем мы вызываем функцию для поиска всех независимых вершин в графе, в которую передаем номер вершины, с которой начинаем обход, кол-во вершин в графе и переменную типа set, созданную ранее. После окончания работы функции вызываем функцию, которая печатает на экране все найденные независимые вершины графа (рисунок 7).

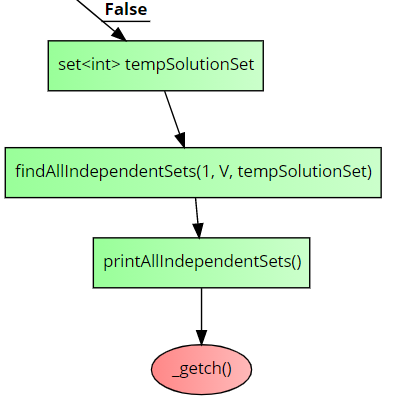


Рисунок 6 – блок-схема вызова основной функции программы

для поиска всех независимых вершин графа

Рассмотрим работу findAllIndependentSets – основной функции, отыскивающей все независимые вершины графа. Ее алгоритм схож с алгоритмом поиска в глубину и основан на рекурсивном просмотре всех вершин графа. Работа функции начинается с того, что мы передаем в неё номер просматриваемой вершины и кол-во вершин в графе. Затем начинаем цикл для прохода по всем вершинам графа (рисунок 8).

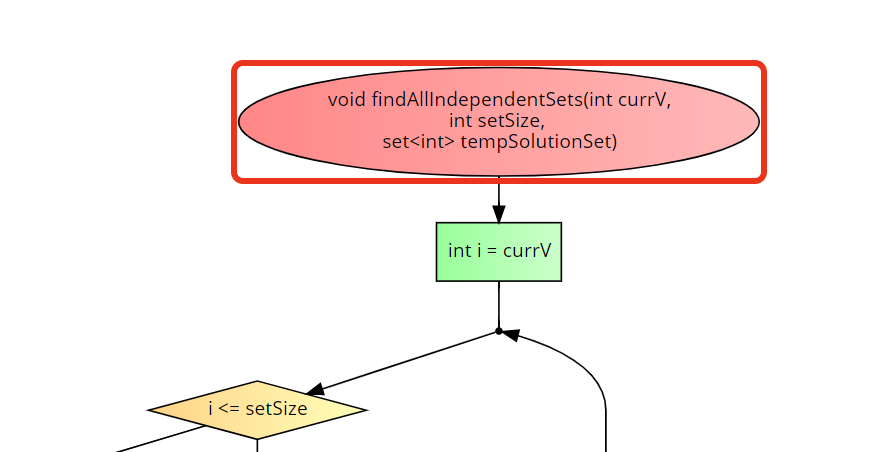


Рисунок 7 – блок-схема начала работы алгоритма с возвратом

Просматривая текущую вершину, мы проверяем для нее все вершины, несмежные с ней. Если очередная вершина несмежна с нашей проверяемой вершиной, мы добавляем ее в множество set (где хранятся уже найденные независимые вершины графа для конкретной вершины) и вызываем функцию рекурсивно функцию поиска независимых вершин уже для нее (рисунок 9).

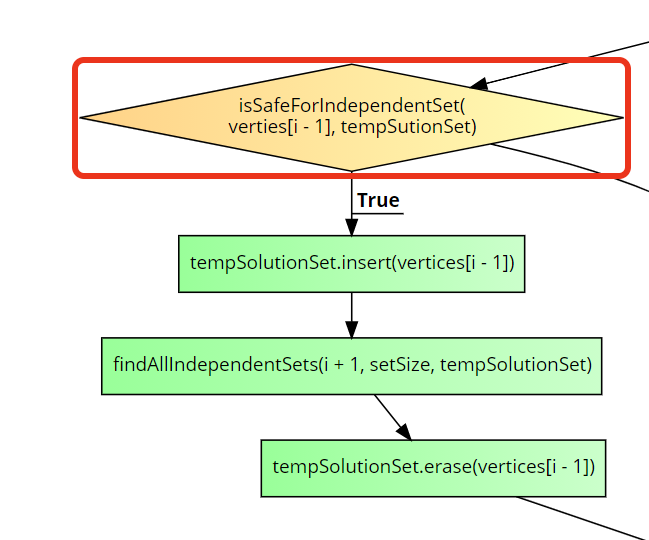


Рисунок 8 – блок-схема рекурсивного вызова алгоритма с возвратом

Таким образом, мы сможем рекурсивно просмотреть все несмежные вершины для нашей основной вершины и записать их в множество. Когда будет найдена последняя такая вершина, мы добавим все найденные несмежные вершины в переменную inpendentSets – она типа set и хранит в себе все независимые множества, найденные в ходе работы алгоритма программы (рисунок 10).

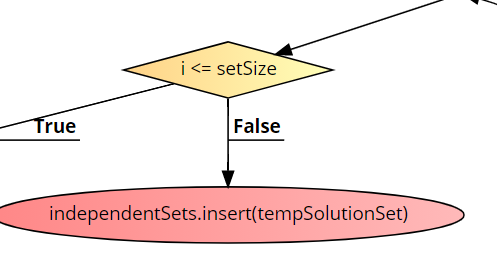


Рисунок 9 – блок-схема добавления найденного множества

независимых вершин в контейнер set

Надо сказать, что на этом шаге будет найдено максимальное множество независимых вершин для просматриваемой вершины, после чего произойдет выход из рекурсии на один шаг, будет удалена последняя добавленная вершина и новое множество вновь будет добавлено в контейнер IndependentSets.

Рассмотрим теперь, как происходит проверка очередной вершины на независимость. Мы передаем в функцию уже найденные независимые вершины и делаем по ним проход циклом for (рисунок 11).

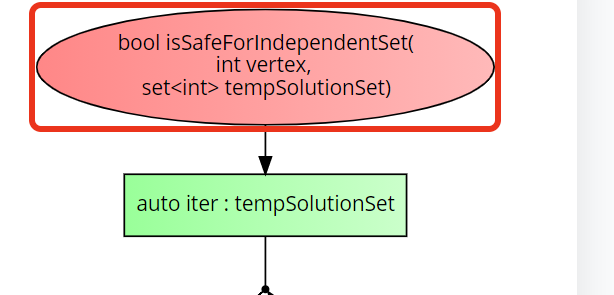


Рисунок 10- блок-схема проверки очередной вершины на независимость

Мы проверяем смежна ли новая вершина с теми вершинами, которые уже хранятся в контейнере set. Если она смежна хоть с одной из них, программа возвращает False и вершина в контейнера добавлена не будет, иначе она возвращает True (рисунок 12).

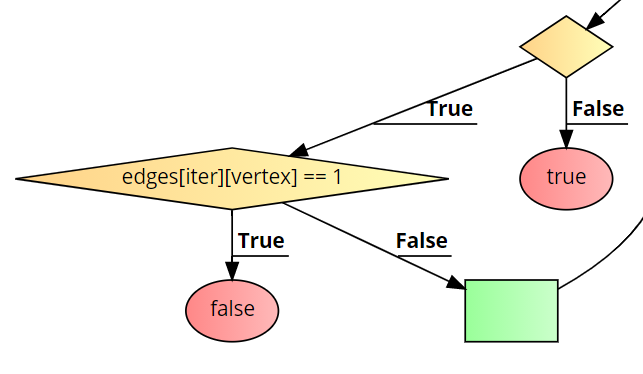


Рисунок 11- блок-схема проверки очередной вершины на независимость

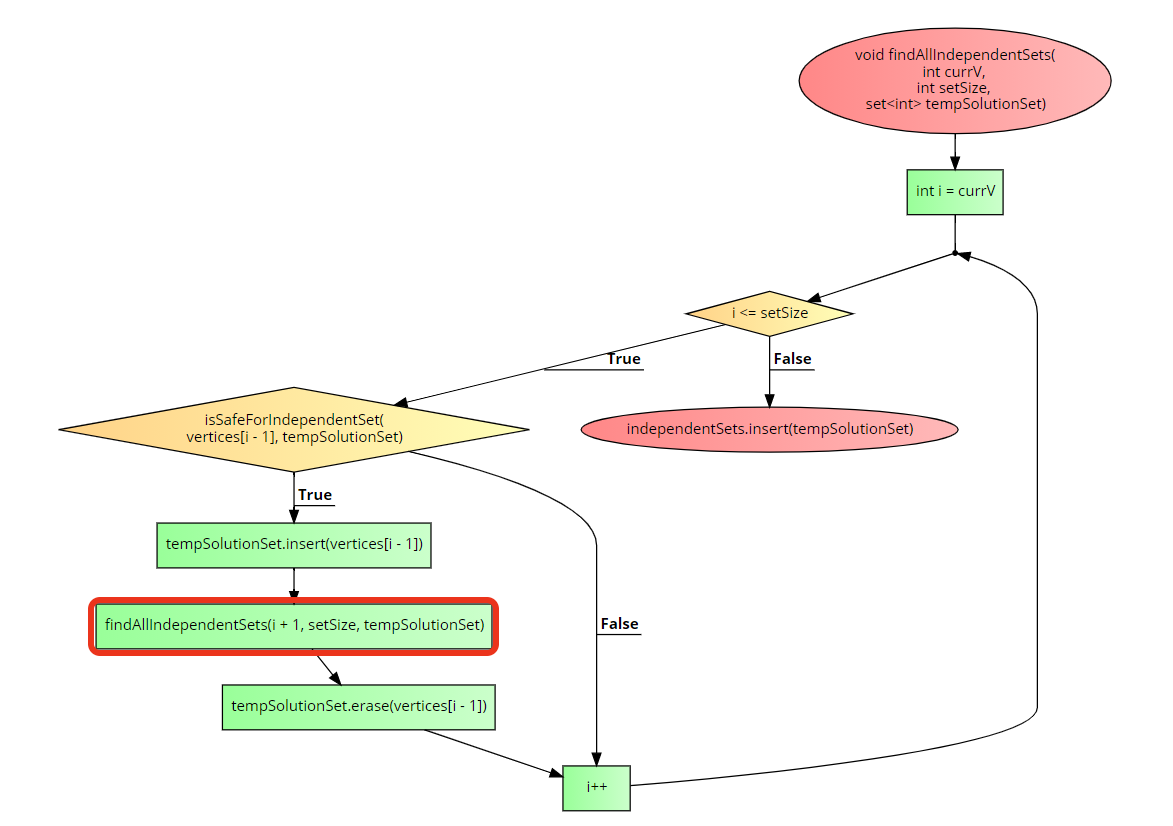


Рисунок 12- блок-схема основного алгоритма программы (поиск с возвратом)

# 5. Описание программы

## 5.1 Общее описание

Задание для курсового проектирования заключается в создании своего программного продукта, наглядно представить все возможные независимые множества вершин графа.

Программа работает в консольном режиме.

В программе возможен ввод размера матрицы смежности с клавиатуры, а также генерация матрицы смежности по заданному количеству вершин.

## 5.2 Функции программы

Для удобства и простоты пользования программой, а также для обеспечения наибольшего быстродействия разделим программу на 3 функции. Листинг программы представлен в приложении А.

**Функция «**Menu()**»**

Функция Menu() отображает консольное меню программы. В данной функции мы можем выбрать 3 вида действий – выбрать размер матрицы смежности (представляющей собой отображение невзвешенного неориентированного графа) вручную, сгенерировать его автоматически или выйти из программы.

Код функции:

int Menu(void)

{

system("CLS");

int c = 0;

setlocale(LC\_ALL, "Rus");

while (c < '0' || c > '2')

{

printf(

"::====================================================::\n"

"|| 1 : Задать размер матрицы автоматически ||\n"

"|| 2 : Задать размер матрицы вручную. ||\n"

"|| 0 : Выход ||\n"

"::====================================================::\n"

"> ");

c = \_getch();

system("cls");

}

return c;

}

**Функция «**Main()**»**

Функция Main() обрабатывает данные, введенные пользователем. Она осуществляет взаимодействие между функциями Menu() и MakeArray() с помощью конструкции switch/case.

Код функции:

void main(void)

{

int Select;

while ((Select = Menu()) != '0' && Select != 27)

switch (Select)

{

case '1':

MakeaArray(1);

break;

case '2':

MakeaArray(2);

break;

}

}

**Функция «**MakeArray()**»**

Функция MakeArray() генерирует матрицу смежности. В неё из функции Menu() передаётся параметр k, если он равен единице, размер матрицы смежности (это определяется количеством вершин в графе) задается автоматически, иначе пользователь вводит его вручную. Размер матрицы должен быть не более 10. Затем на основе заданного размера происходит генерация матрицы смежности. Из данной функции вызывается функция findAllIndependentSets() для поиска всех множеств независимых вершин в графе, после окончания ее работы вызывается функция PrintAllIndependentSets() для вывода всех найденных множеств на экран.

Код функции:

void MakeaArray(int k)

{

setlocale(LC\_ALL, "RUS");

int V;

if (k == 1) {

V = rand() % 9;

}

else {

while (true)

{

printf("\nУкажите размер матрицы V\*V (не более 9): ");

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

if ((V<10) && (V>0)) {

break;

}

else {

printf("Неверный размер матрицы");

}

}

}

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

vertices.push\_back(i);

edges = (int\*\*)malloc(V \* sizeof(int\*));

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

{

edges[i] = (int\*)malloc(V \* sizeof(int));

}

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

{

edges[i][i] = 0;

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

{

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

edges[j][i] = edges[i][j];

}

}

for (int i = 0; i < V; i++) { // вывод матриц смежности

printf("\n");

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

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

}

}

set<int> tempSolutionSet;

findAllIndependentSets(1, V, tempSolutionSet);

printAllIndependentSets();

\_getch();

}

**Функция «**findAllIndependentSets()**»**

Функция findAllIndependentSets () - главная функция программы. В данную функцию передается вершина, с которой мы начинаем обход нашей матрицы, количество всех вершин в графе, а также контейнер set для временного хранения найденных множеств независимых вершин графа. После окончания работы алгоритмы данной функции она должна записать в переменную IndependentSets типа Set все найденные множества независимых вершин графа.

Код функции:

void findAllIndependentSets(int currV, int setSize, set<int> tempSolutionSet)

{

for (int i = currV; i <= setSize; i++) {

if (isSafeForIndependentSet(vertices[i - 1], tempSolutionSet))

{

tempSolutionSet.insert(vertices[i - 1]);

findAllIndependentSets(i + 1, setSize, tempSolutionSet);

tempSolutionSet.erase(vertices[i - 1]);

}

}

independentSets.insert(tempSolutionSet);

}

**Функция «**isSafeForIndependentSet()**»**

Функция isSafeForIndependentSet()– булева функция. Получается на вход номер очередной просматриваемой вершины графа и множество, хранящее уже найденные независимые вершины. Возвращает False, если очередная вершина, данная на проверку, не смежна с уже проверенными вершинами, иначе возвращает True.

Код функции:

bool isSafeForIndependentSet(int vertex, set<int> tempSolutionSet)

{

for (auto iter : tempSolutionSet) {

if (edges[iter][vertex] == 1) {

return false;

}

}

return true;

}

**Функция «**printAllIndependentsSet()**»**

Функция printAllIndependentsSet ()– печатает все найденные независимые вершины, а также параллельно записывает их в файл. Для работы данной функции была подключена библиотека C++ fstream.

Код функции:

void printAllIndependentSets()

{

fstream inOut;

inOut.open("file.txt", ios::out);

printf("\n");

cout << "Все независимые множества вершин в графе: ";

inOut << "Все независимые множества вершин в графе: " << endl;

for (auto iter : independentSets) {

printf("\n");

cout << "{ ";

inOut << "{ ";

for (auto iter2 : iter) {

cout << iter2 + 1 << " ";

inOut << iter2 + 1 << " ";

}

cout << "}";

inOut << "} " << endl;

}

inOut.close();

cout << endl;

independentSets.clear();

}

# Тестирование

Среда разработки MicrosoftVisualStudio 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.

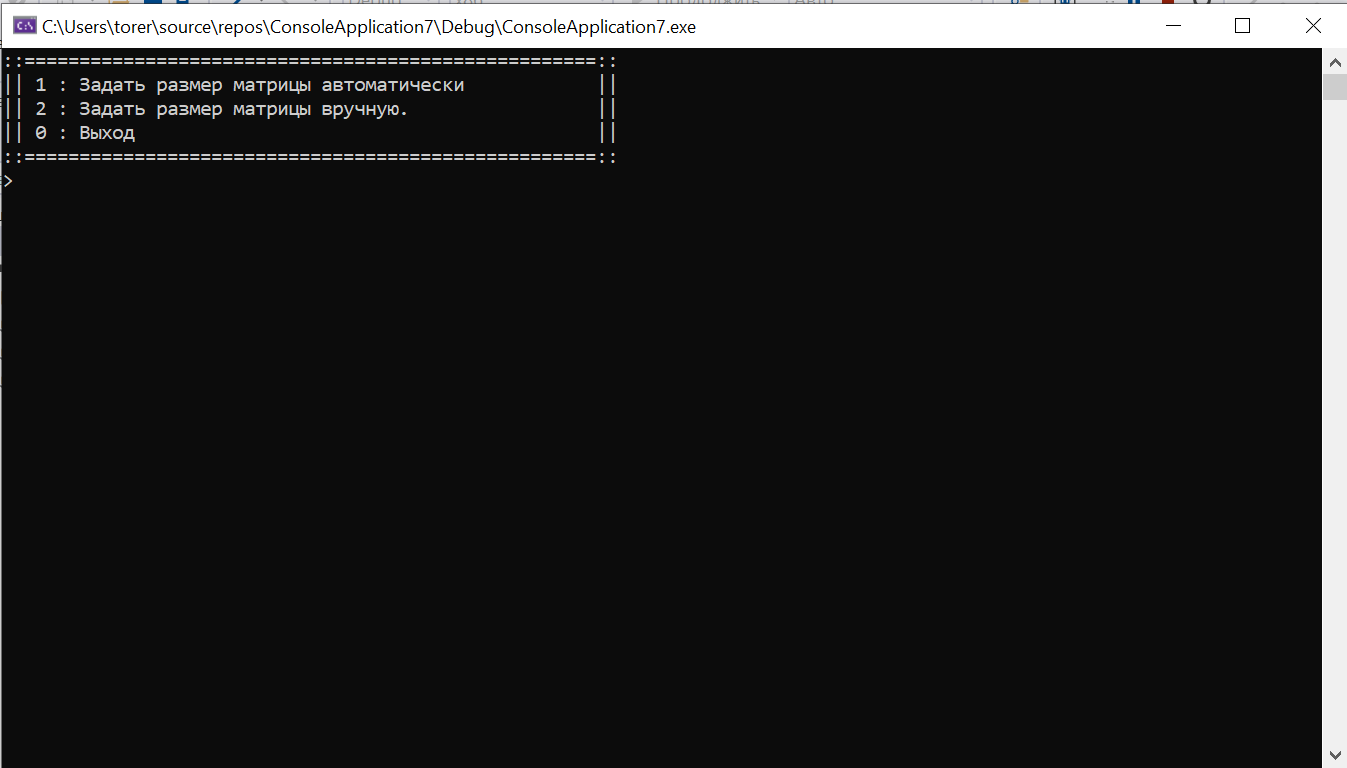
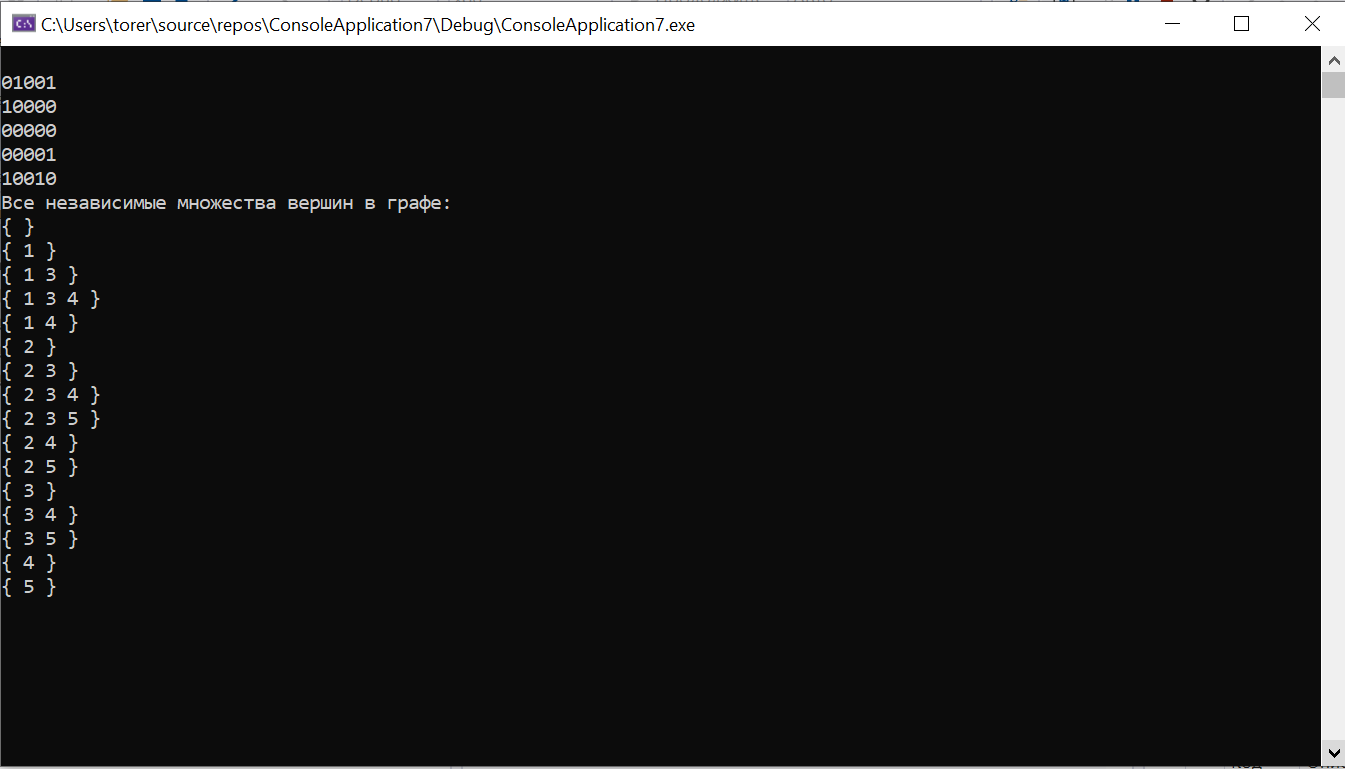
Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин вручную или генерации размера матрица автоматически, а также проверка записи результата программы в файл.

Рисунок 1 – Запуск программы



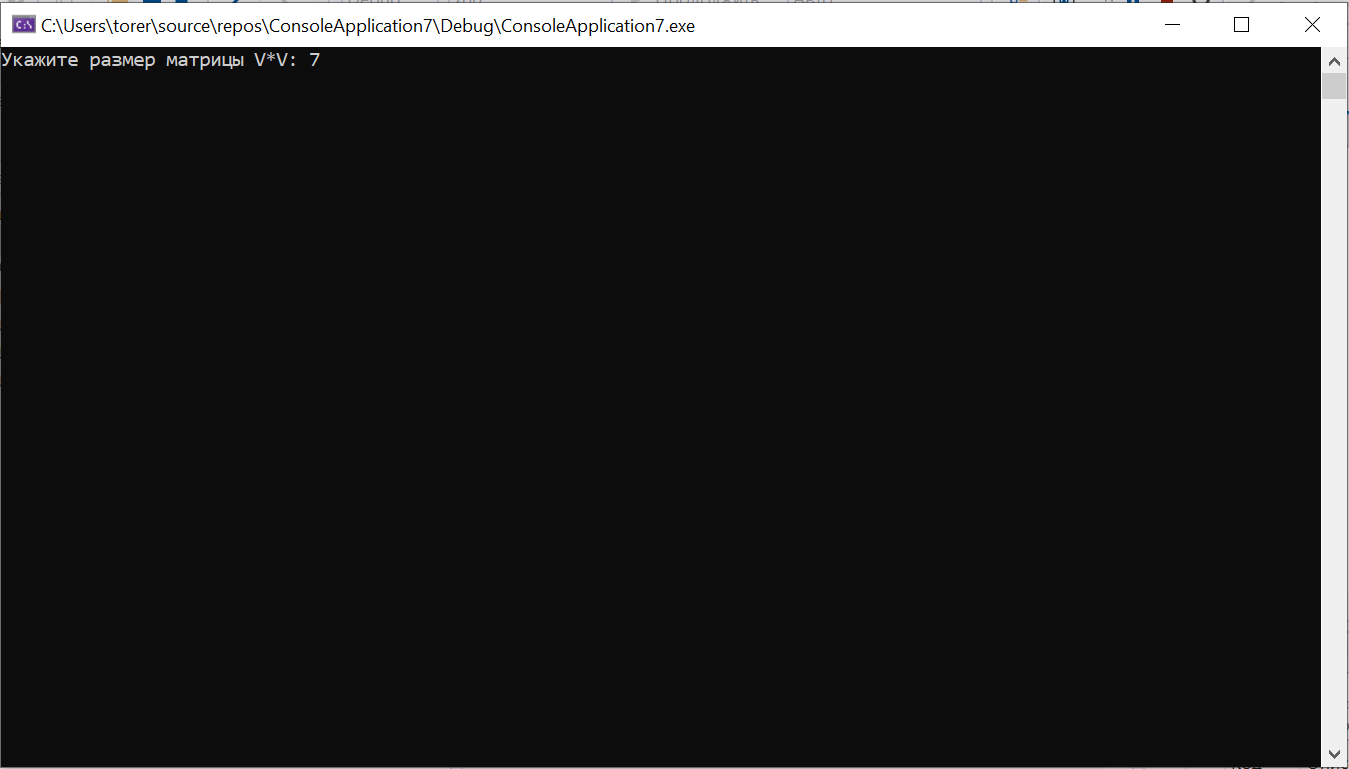
Рисунок 2 – Автоматическая генерация матрицы смежности

Рисунок 3 – Задаем размер матрицы с клавиатуры

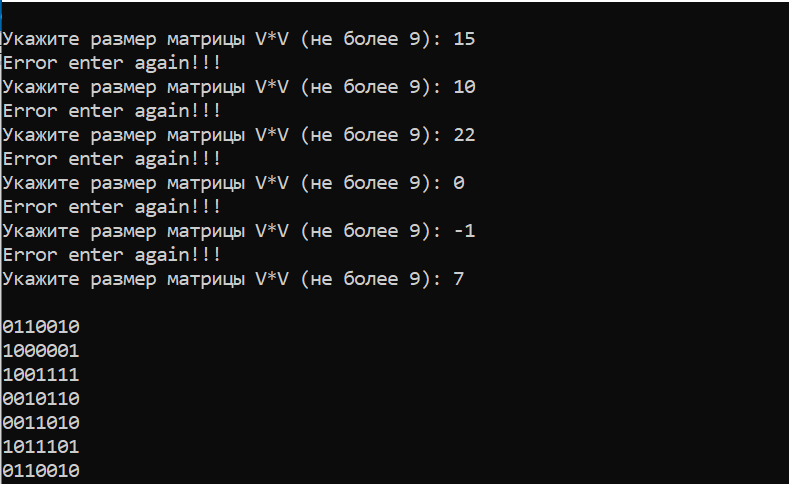


Рисунок 4 – пример ручного ввода с некорректными значениями

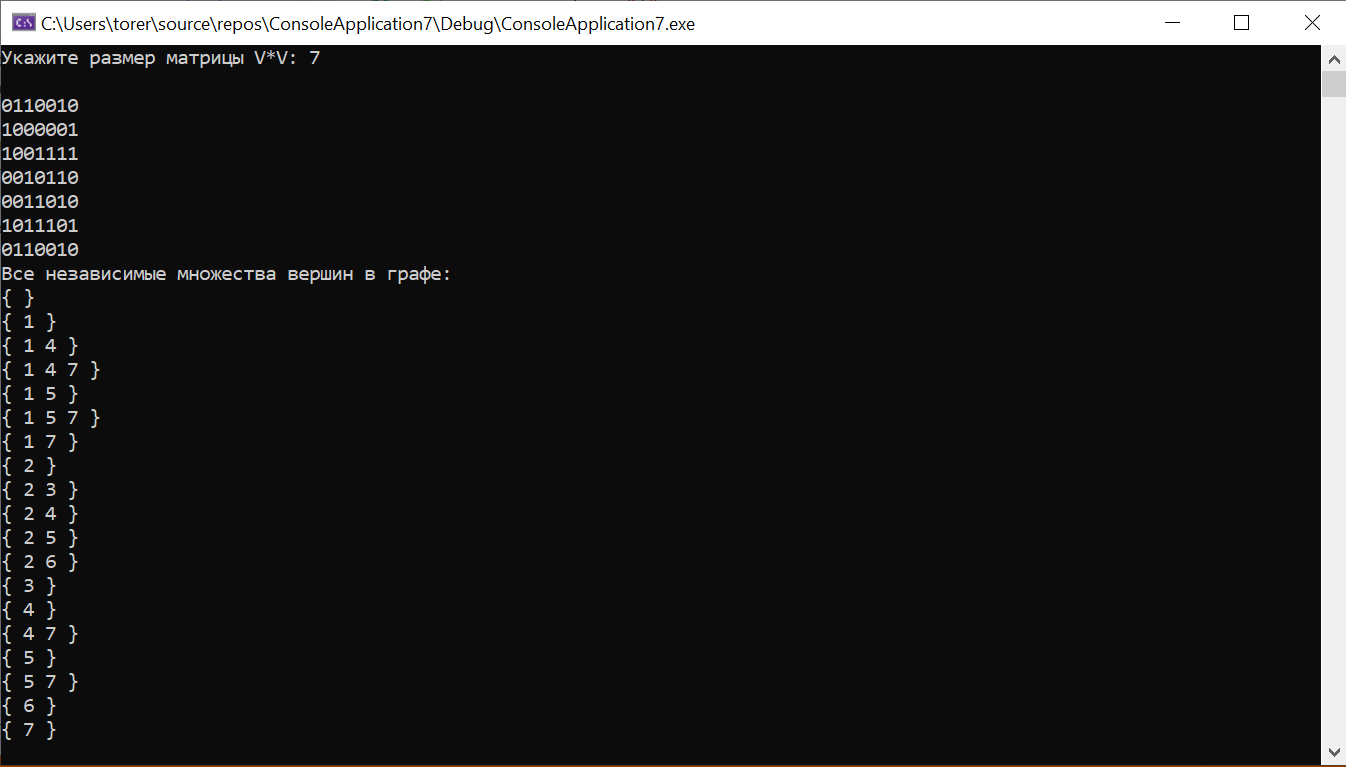


Рисунок 5– Результат работы матрицы смежности при вводе с клавиатуры

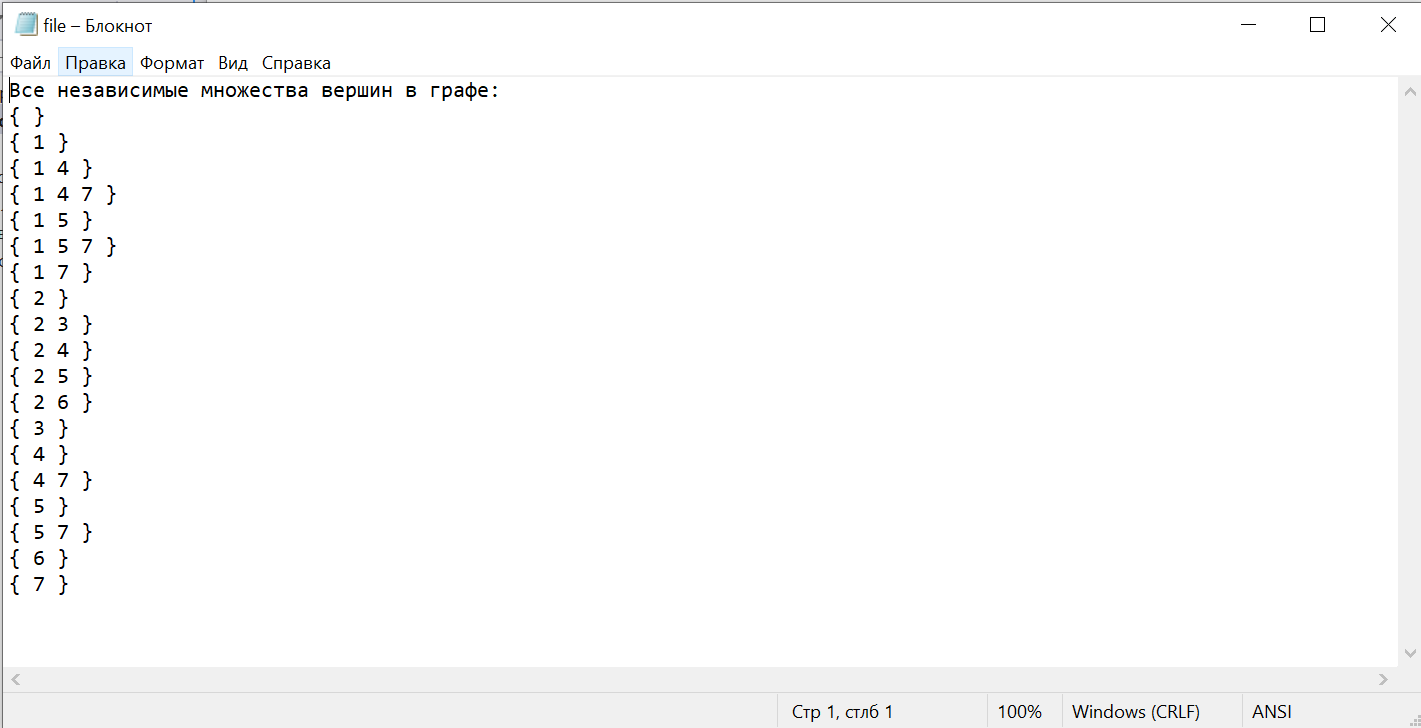


Рисунок 6 – Вывод результата работы программы в файл

Таблица 1 - Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Вывод главного меню | Верно |
| Выбор способа задания размера графа | Вывод сообщения о выборе: задать размер матрицы вручную или выбрать размер автоматически | Верно |
| Ручной ввод | Размер матрицы задается вручную | Верно |
| Ошибка при вводе |  | Верно |
| Автоматическая генерация матрицы | Моментальный вывод матрицы на экран | Верно |
| Сохранение результата работы программы | Результат работы программы сохраняется в текстовой файл | Верно |

# Ручной расчет задачи

Для ручного расчёта решим задачу. Представим граф с 4 вершинами. Выбранный граф изображен на рисунке 19.

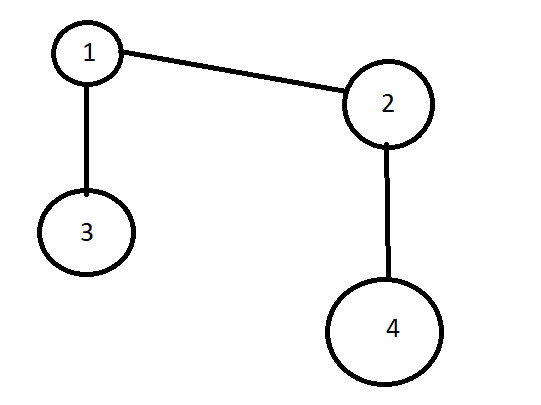


Рисунок 1 – заданный граф для тестирования

1. Задаем пустое множество вершин, которое тоже является независимым.
2. Проверяем первую вершину на смежность. Так как первая вершина не смежна ни с одно вершиной из текущего множества, то добавляем ее в множество{1}.Получается независимое множество
3. Возьмем вершину 2. Проверим ее на смежность с элементами множества. Вершина 2 смежна с вершиной 1 из множества {1}.
4. Берем вершину 3.Проверим ее на смежность с элементами множества. Вершина 3 смежна с вершиной 1 из множества {1}.
5. Берем вершину 4. Она не смежна ни с одной вершиной множества {1}, добавляем ее в множество. Получаем множество {1,4}.
6. Номер текущей вершины больше, чем количество вершин в множестве.
7. Теперь на каждом выходе из рекурсии мы сначала добавляем найденное множество в структуру set, а затем удаляем последний элемент из данного множества.
8. Возвращаемся на предыдущий уровень.
9. Берем следующую вершину. Это вершина 2. Проверяем ее на смежность с элементами множества. Так как множество пустое, записываем ее в множество. Получаем множество {2}.
10. Проверяем вершину 3 на смежность с элементами множества, вершина не смежна, записываем ее в множество. Получаем {2,3}.
11. Проверяем вершину 4. Вершину 4 смежна с вершиной 2 из множества {2,3}.
12. Номер текущей вершины больше, чем количество вершин в множестве.
13. Возвращаемся на предыдущий уровень.
14. Берем следующую вершину. Это вершина 3. Проверяем ее на смежность с элементами множества. Так как множество пустое, записываем ее в множество. Получаем множество {3}.
15. Проверяем вершину 4 на смежность с элементами множества, вершина не смежна, записываем ее в множество. Получаем {3,4}.
16. Номер текущей вершины больше, чем количество вершин в множестве.
17. Проверяем вершину 4 на смежность с элементами множества, вершина не смежна, записываем ее в множество. Получаем {4}.
18. Поиск завершен.

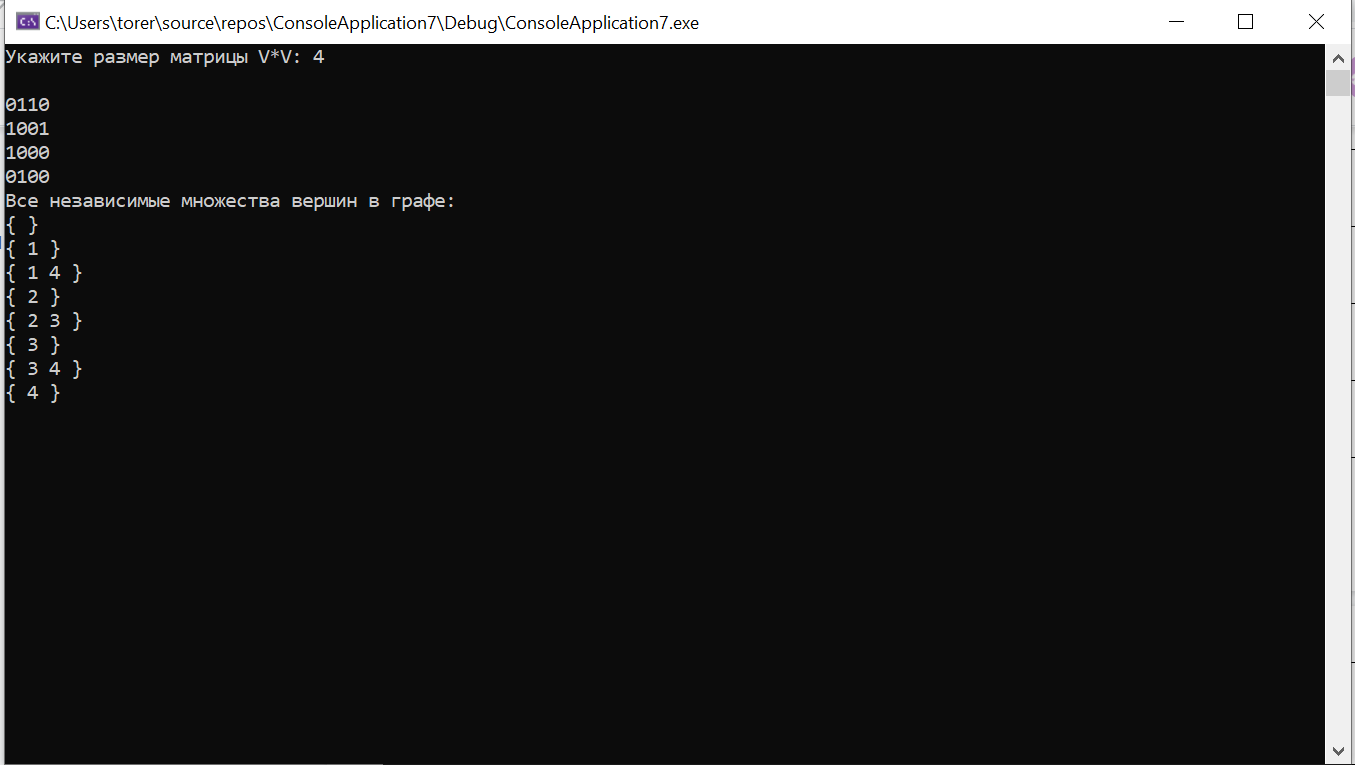
Проверим программой. Результат работы показан на рисунке 7.Таким образом, можно сделать вывод, что программа работает верно.

Рисунок 2 – Тестирование работы программы

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска всех независимых множеств вершин в графе, представленном в матрице смежности в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежностей и работы с ними. Приобретены навыки по осуществлению рекурсивного прохода по вершинам графа. Углублены знания языка программирования Cи++.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Новиков Ф.А. «Дискретная математика для программистов» 2-е изд. – СПб.: Питер, 2000.
2. Электронныйресурс: <https://www.intuit.ru/studies/courses/648/504/lecture/11464>

Дата обращения: 01.11.2017.

1. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006 5. Х. М. Дейтел, П. Дж. Дейтел «Как программировать на C++» - Бином-Пресс, 2008.
2. М. Эллис, Б. Строуструп. Справочное руководство по языку C++ с комментариями: Пер. с англ. - Москва: Мир, 1992. 445с.
3. Х. Дейтел, П. Дейтел. Как программировать на C++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. 1024с.

# Приложение А.Листинг программы.

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <set>

#include <map>

#include <vector>

#include <fstream>

using namespace std;

set<set<int> > independentSets;

int\*\* edges;

vector<int> vertices;

void printAllIndependentSets()

{

fstream inOut;

inOut.open("file.txt", ios::out);

printf("\n");

cout << "Все независимые множества вершин в графе: ";

inOut << "Все независимые множества вершин в графе: " << endl;

for (auto iter : independentSets) {

printf("\n");

cout << "{ ";

inOut << "{ ";

for (auto iter2 : iter) {

cout << iter2 + 1 << " ";

inOut << iter2 + 1 << " ";

}

cout << "}";

inOut << "} " << endl;

}

inOut.close();

cout << endl;

independentSets.clear();

}

bool isSafeForIndependentSet(int vertex, set<int> tempSolutionSet)

{

for (auto iter : tempSolutionSet) {

if (edges[iter][vertex] == 1) {

return false;

}

}

return true;

}

void findAllIndependentSets(int currV, int setSize, set<int> tempSolutionSet)

{

for (int i = currV; i <= setSize; i++) {

if (isSafeForIndependentSet(vertices[i - 1], tempSolutionSet))

{

tempSolutionSet.insert(vertices[i - 1]);

findAllIndependentSets(i + 1, setSize, tempSolutionSet);

tempSolutionSet.erase(vertices[i - 1]);

}

}

independentSets.insert(tempSolutionSet);

}

void MakeaArray(int k)

{

setlocale(LC\_ALL, "RUS");

int V;

if (k == 1) {

V = rand() % 9;

}

else {

while (true)

{

printf("\nУкажите размер матрицы V\*V (не более 9): ");

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

if ((V<10) && (V>0)) {

break;

}

else {

printf("Error enter again!!!");

}

}

}

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

vertices.push\_back(i);

edges = (int\*\*)malloc(V \* sizeof(int\*));

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

{

edges[i] = (int\*)malloc(V \* sizeof(int));

}

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

{

edges[i][i] = 0;

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

{

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

edges[j][i] = edges[i][j];

}

}

for (int i = 0; i < V; i++) { // вывод матриц смежности

printf("\n");

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

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

}

}

set<int> tempSolutionSet;

findAllIndependentSets(1, V, tempSolutionSet);

printAllIndependentSets();

\_getch();

}

int Menu(void)

{

system("CLS");

int c = 0;

setlocale(LC\_ALL, "Rus");

while (c < '0' || c > '2')

{

printf(

"::====================================================::\n"

"|| 1 : Задать размер матрицы автоматически ||\n"

"|| 2 : Задать размер матрицы вручную. ||\n"

"|| 0 : Выход ||\n"

"::====================================================::\n"

"> ");

c = \_getch();

system("cls");

}

return c;

}

void main(void)

{

int Select;

while ((Select = Menu()) != '0' && Select != 27)

switch (Select)

{

case '1':

MakeaArray(1);

break;

case '2':

MakeaArray(2);

break;

}

}