Министерство просвещения Российской Федерации  
Кубанский Государственный Университет

Отчёт

по выполнению индивидуального задания № 1  
по дисциплине «Конструирование алгоритмов и структур данных».

Выполнила студентка группы 16/2  
Шишкина Диана Олеговна  
/

/

Краснодар

2020

**Вариант № 11**

**Задание 1.**

Дан неориентированный граф. Известна наибольшая клика. Найти наименьшее покрытие его алгебраического дополнения.

1. Математическая постановка задачи

Кликой в неориентированном графе называется подмножество вершин , такое что для любых двух вершин в существует ребро, их соединяющее. Это эквивалентно утверждению, что подграф, порождённый , является полным.

Множество вершин графа называется независимым, если никакие две вершины этого множества не соединены ребром.

Дополнение графа (обратный граф) — граф , имеющий то же множество вершин, что и заданный граф , но в котором две несовпадающие вершины смежны тогда и только тогда, когда они не смежны в .

Множество вершин независимо тогда и только тогда, когда оно является кликой в дополнении графа.

Вершинное покрытие для неориентированного графа — это множество его вершин , такое, что, у каждого ребра графа хотя бы один из концов входит в вершину из .

Множество вершин независимо тогда и только тогда, когда его дополнение является вершинным покрытием.

1. Описание алгоритма решения

Пусть множество – множество вершин исходного графа , а его подмножество – клика. Рассмотрим граф - дополнение исходного графа . Так как множество является кликой в исходном графе , то в графе оно независимо. Дополнение множества множество является вершинным покрытием в графе и, если – наибольшая клика графа , то - наименьшее вершинное покрытие графа .

Работа алгоритма заключается в переборе всех вершин исходного графа. Для каждой вершины устанавливается ее принадлежность клике. Если вершина не принадлежит клике, то она включается в решение, если нет, то алгоритм пропускает ее.

1. Техническое описание программного продукта

Алгоритм реализован на языке программирования C#. Применены заголовочные файлы using System, using System.Collections.Generic, using System.Windows.Forms.

Для решения задач были реализованы следующие классы: Vertex, Edge и Graph.

Класс Vertex содержит:

* public string Name – свойство, определяющее имя вершины;
* public List<Vertex> Adjacent = new List<Vertex>() – список вершин, смежных с данной;
* public Vertex(string name) => Name = name – конструктор, позволяющий сразу задать имя вершины;
* public Vertex() { } – конструктор.

Класс Edge содержит:

* public int Weight { get; set; } – свойство, определяющее вес ребра;
* public Vertex Begin { get; private set; } – свойство, определяющее начало ребра;
* public Vertex End { get; private set; } – свойство, определяющее конец ребра;
* public Edge(Vertex begin, Vertex end) – конструктор, связывающий вершины begin и end (вершина end добавляется в список смежных вершин вершины begin).

Класс Graph содержит:

* public Graph() – конструктор;
* public List<Vertex> Vertices – список вершин;
* public List<Edge> Edges – список ребер;
* public bool isDirected { get; private set; } – свойство логического типа, определяющее ориентированность графа (false – граф неориентированный, true – граф ориентированный);
* public bool areConnected(Vertex vertex, Vertex adjacent) – функция, определяющая смежность вершины vertex с вершиной adjacent посредством поиска ребра (vertex; adjacent) в списке ребер текущего графа (false – vertex несмежна с adjacent, true – смежна);
* public Edge GetEdge(Vertex Begin, Vertex End) – функция, возвращающая ребро (Begin; End). Возвращает null, если данное ребро не принадлежит графу;
* public void EnterAdjacencyMatrix(string str) – функция ввода графа, представленном матрицей смежности. Матрица записана в строковую константу. Для разделения столбцов используется символ “перевод строки” (символ ‘\n’). В зависимости от полученных данных устанавливается значение свойства isDirected;
* public List<Vertex> VertexAddition(List<Vertex> vertices) – функция, возвращающая дополнение заданного множества вершин. Множество вершин и его дополнение задаются списком вершин.

1. Инструкция по эксплуатации

После запуска программы появляется главное меню (рис. 1).

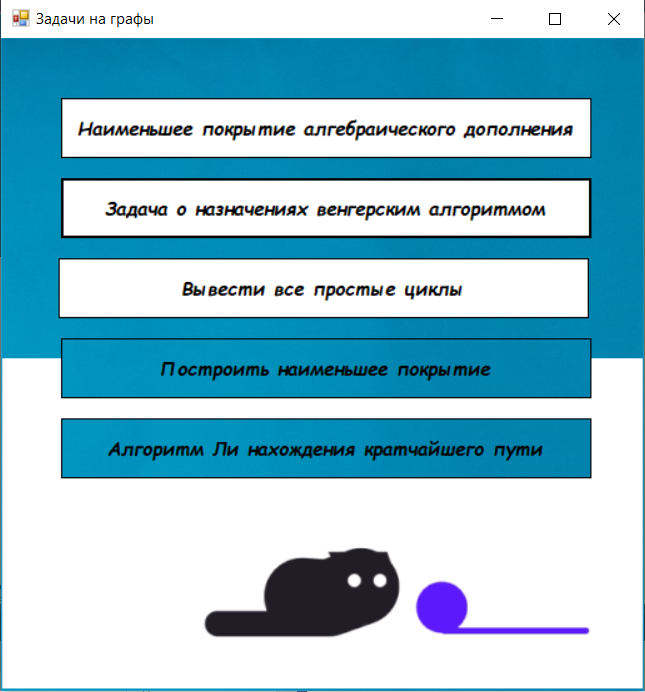


Рисунок 1. Меню программы

Далее следует нажать кнопку с названием интересующей нас задачи. В данном случае это кнопка «Наименьшее покрытие алгебраического дополнения».

Попадаем в следующее окно (рис. 2).

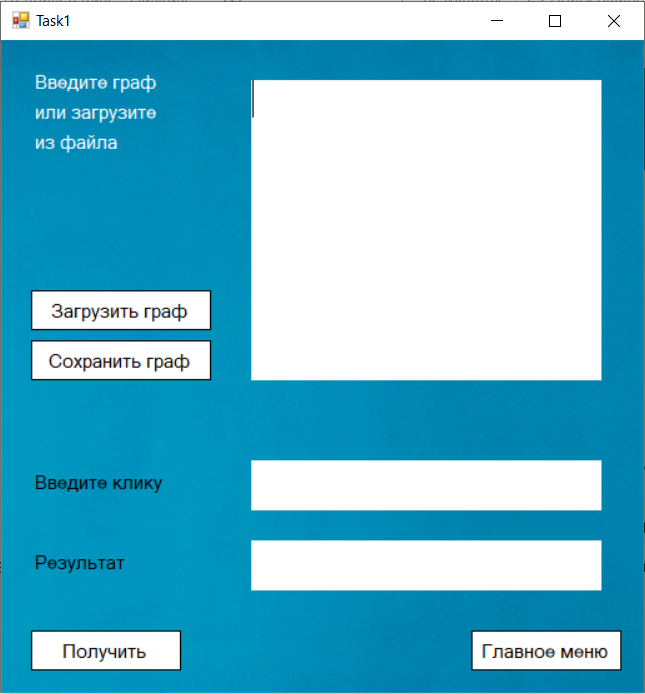


Рисунок 2. Задача 1

В большем поле нужно ввести граф. Граф задается матрицей смежности. Кнопка «Загрузить граф» используется для ввода графа из существующего файла формата txt. Кнопка «Сохранить граф», соответственно, для сохранения введенного графа в txt-файл. Кнопка «Получить» запускает алгоритм и выводит результат.

Введем некоторый граф и его наибольшую клику. Нажимаем «Получить». Результат работы представлен на рисунке 3.

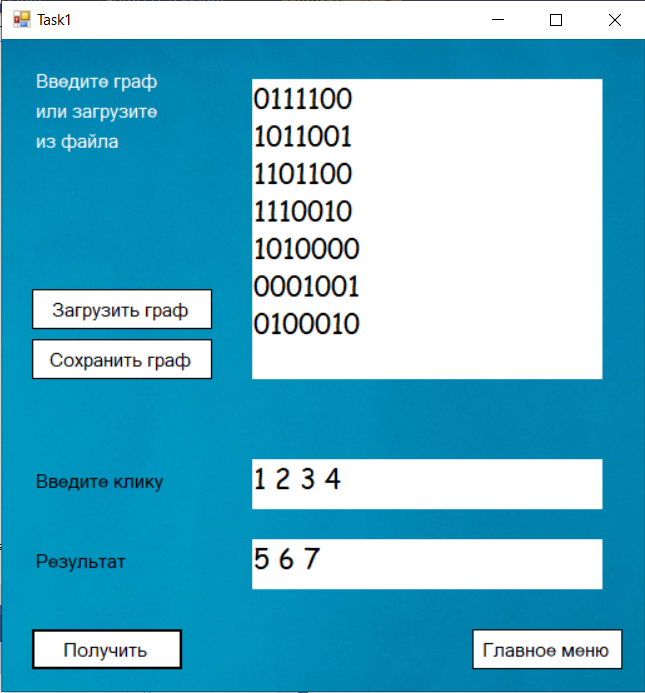
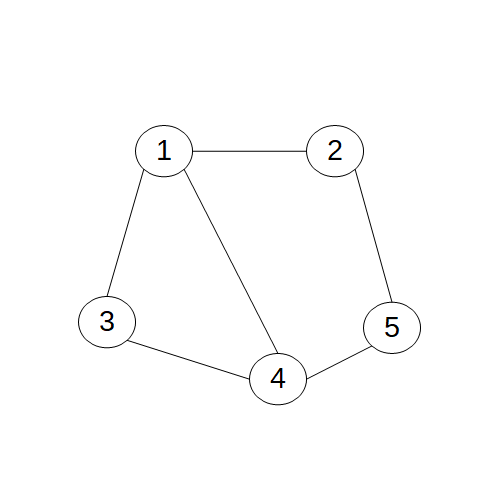
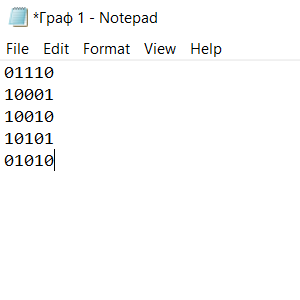


Рисунок 3. Результат работы задачи 1

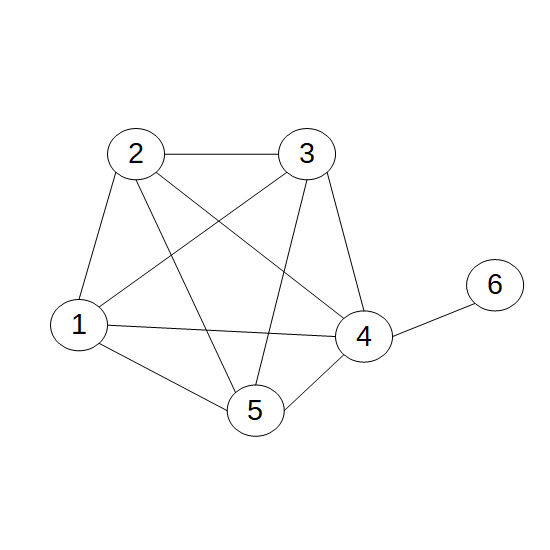
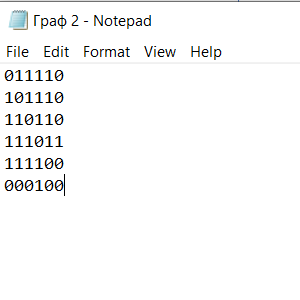
1. Набор графов для тестирования

Граф 1

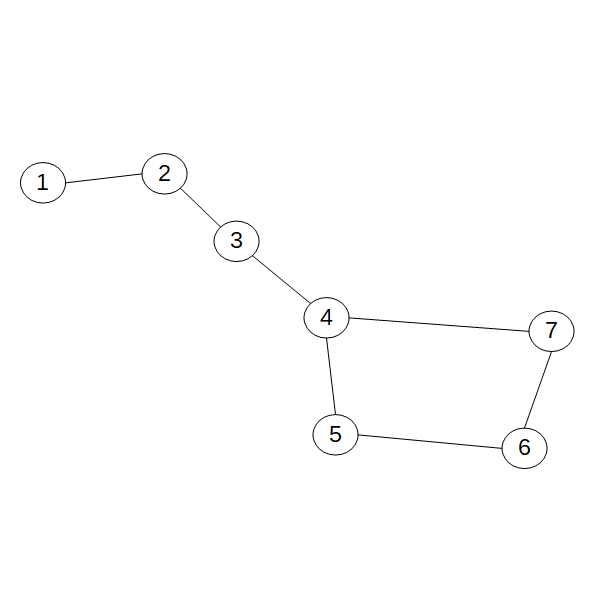
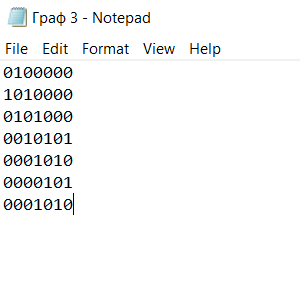
Клика: 1, 3, 4. Результат: 2, 5.

Граф 2

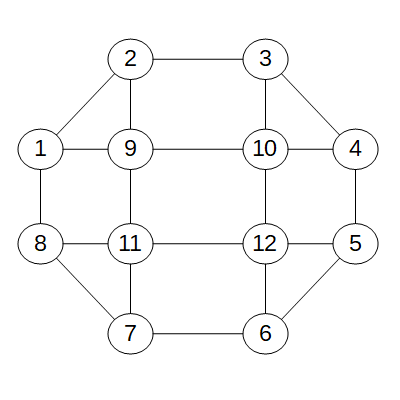
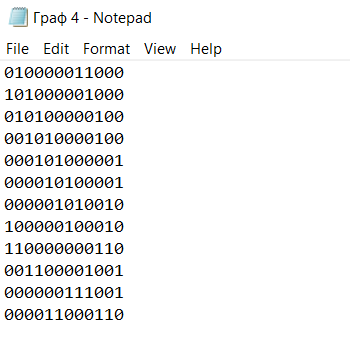
Клика: 1, 2, 3, 4, 5. Результат: 6.

Граф 3

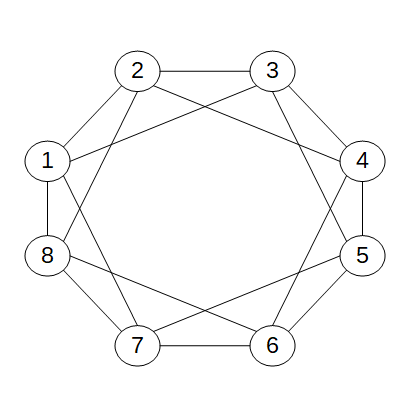
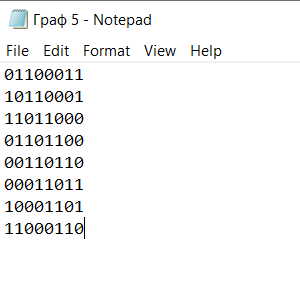
Клика: 4, 5. Результат: 1, 2, 3, 6, 7.

Граф 4

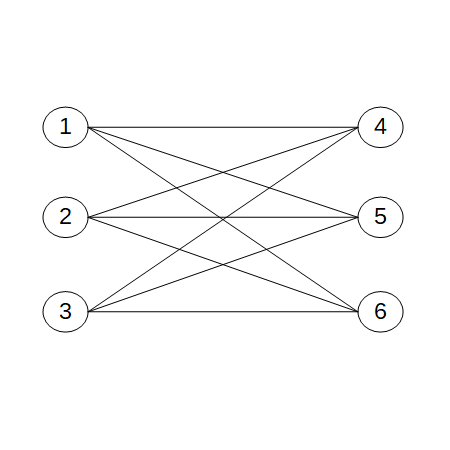
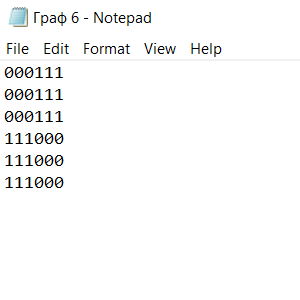
Клика: 7, 8, 11. Результат: 1, 2, 3, 4, 5, 6, 9, 10, 12.

Граф 5

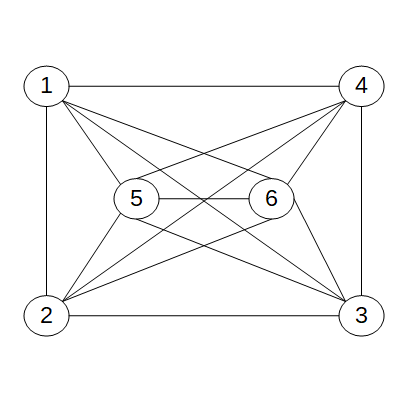
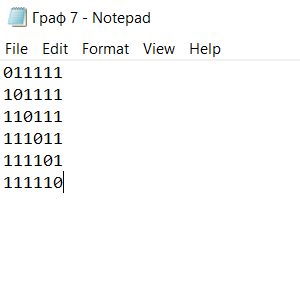
 

Клика: 4, 5, 6. Результат: 1, 2, 3, 7, 8.

Граф 6

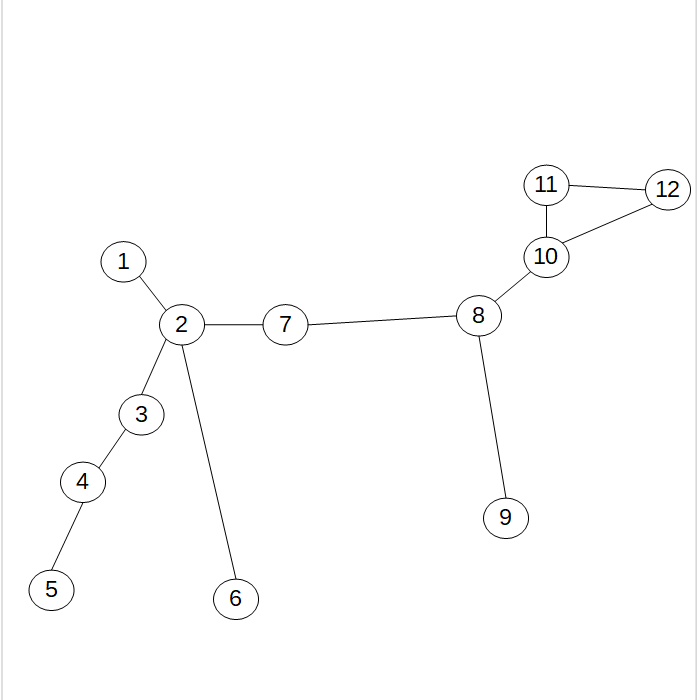
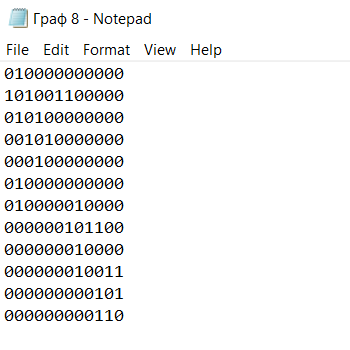
 

Граф 7

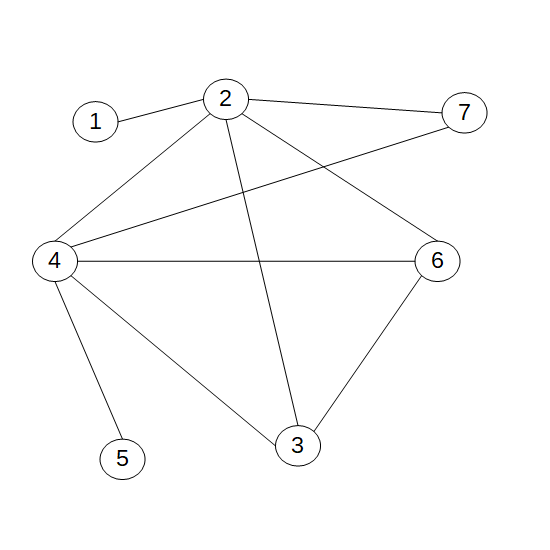
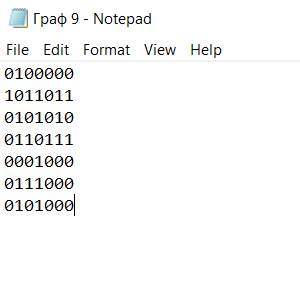
Клика: 1, 2, 3, 4, 5, 6. Результат: не найдено.

Граф 8

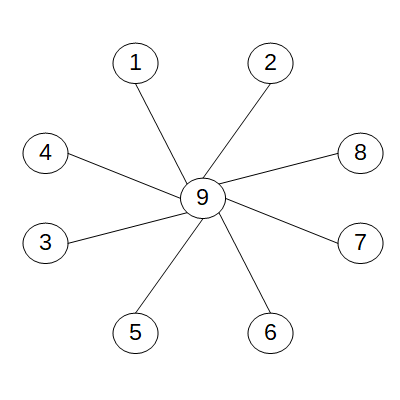
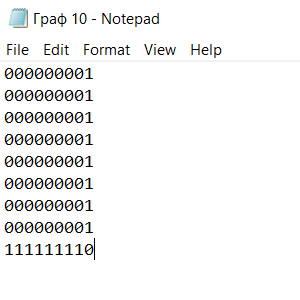
Клика: 10, 11, 12. Результат: 1, 2, 3, 4, 5, 6, 7, 8, 9.

Граф 9

Клика: 2, 3, 4, 6. Результат: 1, 5, 7.

Граф 10

Клика: 1, 9. Результат: 2, 3, 4, 5, 6, 7, 8.

**Задание 2.**

Дана матрица назначений. Решить задачу о назначениях венгерским алгоритмом.

1. Математическая постановка задачи

Неориентированный граф {\displaystyle G=(W,E)} называется двудольным, если множество его вершин можно разбить на две части {\displaystyle U\cup V=W}, так, что ни одна вершина в {\displaystyle U} не соединена с вершинами в {\displaystyle U} и ни одна вершина в {\displaystyle V} не соединена с вершинами в {\displaystyle V}.

Пусть дан граф , паросочетание в — это множество попарно несмежных рёбер, то есть рёбер, не имеющих общих вершин.

Максимальное паросочетание — это такое паросочетание в графе , которое не содержится ни в каком другом паросочетании этого графа, то есть к нему невозможно добавить ни одно ребро, которое бы являлось несмежным ко всем рёбрам паросочетания.

Вершина называется насыщенной, если ей смежно ребро из текущего паросочетания.

Вершина, которой не смежно ни одно ребро из текущего паросочетания, называется ненасыщенной.

Путь нечётной длины, в котором первое ребро не принадлежит паросочетанию, а для всех последующих рёбер происходит чередование (принадлежит/не принадлежит) — называется увеличивающим путём.

1. Описание алгоритма решения

Назовём потенциалом два произвольных массива чисел и таких, что выполняется условие: . Ребро  будем считать жёстким, если выполняется: .

Перейдём непосредственно к описанию алгоритма.

* В начале алгоритма потенциал полагается равным нулю , и паросочетание  полагается пустым.
* Далее, на каждом шаге алгоритм пытается, не меняя потенциала, увеличить мощность текущего паросочетания  на единицу, которое ищется в графе жёстких рёбер . Для этого используется обычный [алгоритм Куна поиска максимального паросочетания в двудольных графах](https://e-maxx.ru/algo/kuhn_matching).

Алгоритм Куна:

Все рёбра паросочетания  ориентируются по направлению от второй доли к первой, все остальные рёбра графа  ориентируются в противоположную сторону. Из всех ненасыщенных вершин первой доли запускается обход в глубину или в ширину. Если в результате обхода удалось достигнуть ненасыщенной вершины второй доли, то это означает, что мы нашли увеличивающий путь из первой доли во вторую. Если прочередовать рёбра вдоль этого пути, то тем самым мы увеличим мощность паросочетания на единицу. Если же увеличивающего пути не было, то это означает, что текущее паросочетание  — максимально в графе , поэтому в таком случае переходим к следующему пункту.

* Если на текущем шаге не удалось увеличить мощность текущего паросочетания, то производится пересчёт потенциала таким образом, чтобы на следующих шагах появилось больше возможностей для увеличения паросочетания.

Обозначим через  множество вершин первой доли, которые были посещены обходом алгоритма Куна при попытке поиска увеличивающей цепи; через  — множество посещённых вершин второй доли.

Посчитаем величину : .Эта величина строго положительна.

Теперь пересчитаем потенциал таким образом: для всех вершин  сделаем , а для всех вершин  — сделаем . Получившийся потенциал по-прежнему останется корректным потенциалом.

Кроме того, старое паросочетание  из жёстких рёбер можно будет оставить, т. е. все рёбра паросочетания останутся жёсткими.

Наконец, чтобы показать, что изменения потенциала не могутпроисходить бесконечно, заметим, что при каждом таком изменении потенциала количество вершин, достижимых обходом, т. е. , строго увеличивается.

Таким образом, всего может происходить не более  пересчётов потенциала, прежде чем обнаружится увеличивающая цепочка и мощность паросочетания  будет увеличена.

Таким образом, рано или поздно будет найден потенциал, которому соответствует совершенное паросочетание , являющееся ответом на задачу.

1. Техническое описание программного продукта

Алгоритм реализован в методе public static int[] HungarianAlgorithm(this int[,] Matrix), который является расширением для двумерного массива.

Метод public static void Fill<T>(this T[] mas, T value) является расширением для одномерных массивов. Присваивает всем элементам массива некоторое значение value.

Данная реализация ищет решение для прямоугольной входной матрицы , где . Матрица хранится в 1-индексации в целях удобства и краткости кода. Дело в том, что в данной реализации вводятся фиктивные нулевая строка и нулевой столбец, что позволяет написать многие циклы в общем виде, без дополнительных проверок.

Массивы  и  хранят потенциал.

Массив  содержит паросочетание: для каждого столбца  он хранит номер соответствующей выбранной строки  (или 0, если пока ничего не выбрано). При этом  для удобства реализации полагается равным номеру текущей рассматриваемой строки.

Массив  содержит для каждого столбца вспомогательные минимумы, необходимые для быстрого пересчёта потенциала:

*.*

Массив  содержит информацию о том, где эти минимумы достигаются, чтобы мы впоследствии смогли восстановить увеличивающую цепочку.

1. Инструкция по эксплуатации

Данной задаче соответствует кнопка «Задача о назначениях венгерским алгоритмом».

В данном форме (рис. 4) имеется таблица, в которую записывается задача. Кнопки «Добавить строку», «Добавить столбец», «Удалить строку», «Удалить столбец» нужны для добавления/удаления строк/столбцов в таблице.

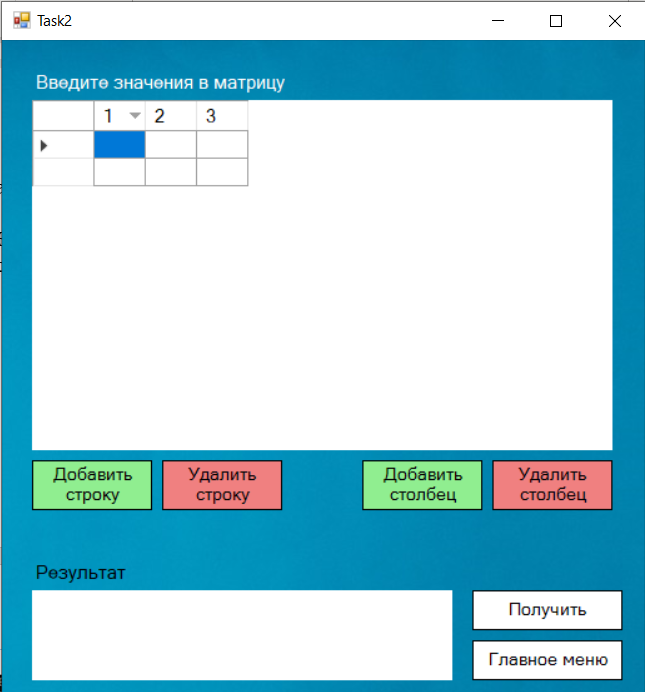


Рисунок 4. Задача 2

Введем в таблицу некоторые значения и нажмем «Получить». Программа выводит (рис. 5) координаты ячеек, сумма которых оптимальна (в данном случае наименьшая).

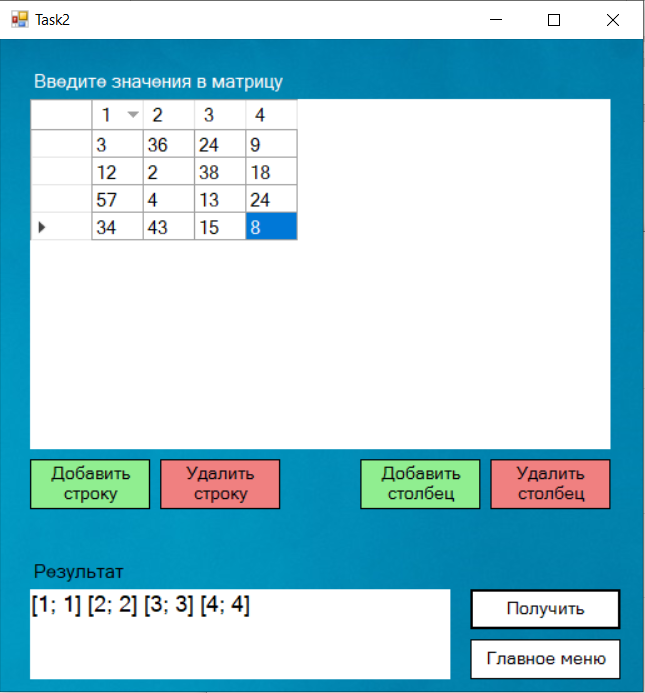
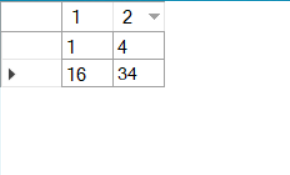


Рисунок 5. Результаты задачи 2

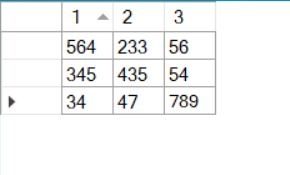
1. Данные для тестирования

Матрица 1



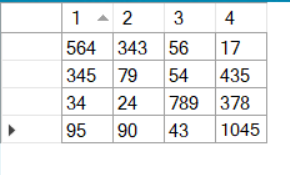
Результат: [1; 2] [2; 1].

Матрица 2



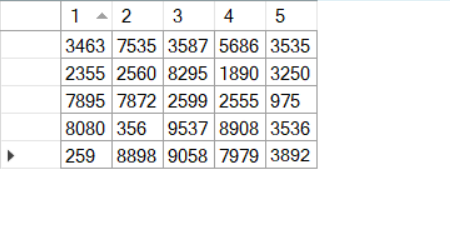
Результат: [1; 2] [2; 3] [3; 1].

Матрица 3



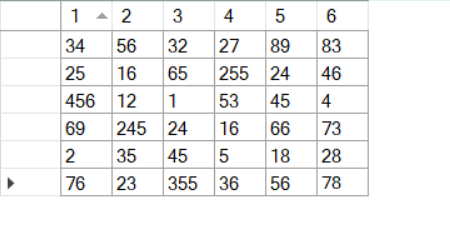
Результат: [1; 4] [2; 2] [3; 1] [4; 3].

Матрица 4



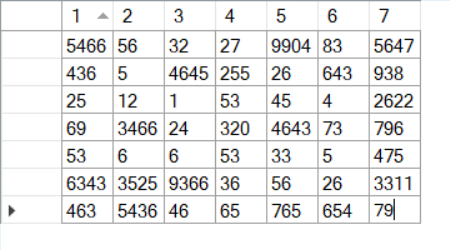
Результат: [1; 3] [2; 4] [3; 5] [4; 2] [5; 1].

Матрица 5



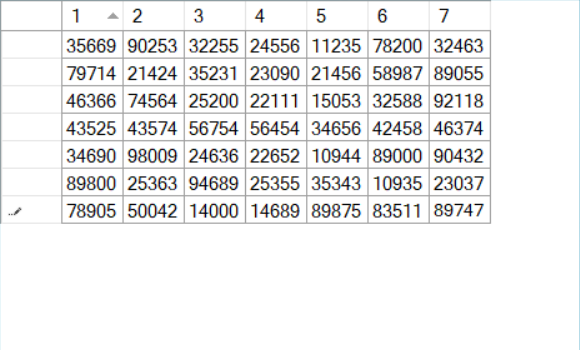
Результат: [1; 3] [2; 5] [3; 6] [4; 4] [5; 1] [6; 2].

Матрица 6



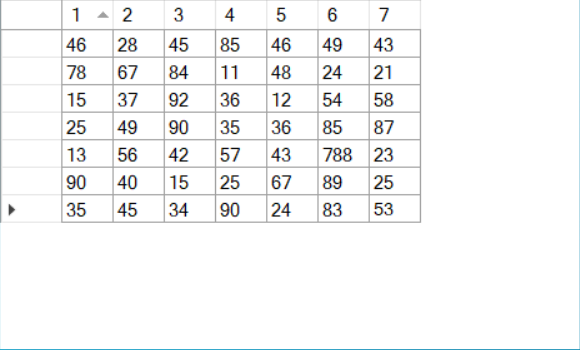
Результат: [1; 4] [2; 5] [3; 1] [4; 3] [5; 2] [6; 6] [7; 7].

Матрица 7



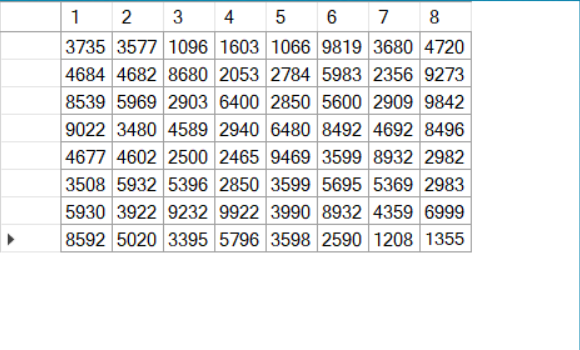
Результат: [1; 7] [2; 2] [3; 4] [4; 1] [5; 5] [6; 6] [7; 3].

Матрица 8



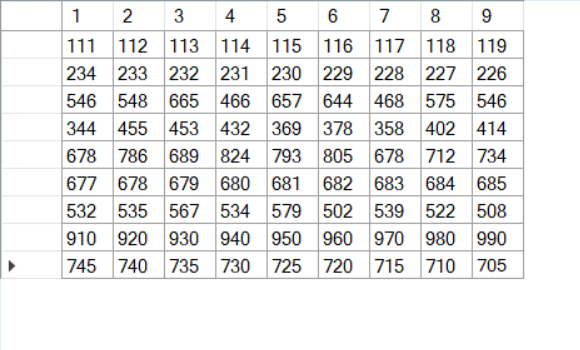
Результат: [1; 2] [2; 6] [3; 1] [4; 4] [5; 7] [6; 3] [7; 5].

Матрица 9



Результат: [1; 3] [2; 7] [3; 5] [4; 4] [5; 6] [6; 1] [7; 2] [8; 8].

Матрица 10



Результат: [1; 5] [2; 8] [3; 4] [4; 1] [5; 7] [6; 3] [7; 6] [8; 2] [9; 9].

**Задание 3.**

Дан неориентированный граф, вывести все простые циклы (без повторяющихся ребер и без повторяющихся вершин).

1. Математическая постановка задачи

Цикл – это последовательность вершин, начинающаяся и заканчивающаяся в одной и той же вершине, и каждые две последовательные вершины в последовательности смежны.

Простой цикл — это цикл без повторного прохода по ребру или посещения вершины дважды, за исключением начальной и конечной вершин*.*

1. Описание алгоритма решения

Рассматривается множество ребер графа. Для каждого ребра алгоритм выполняет следующие действия:

* Текущее ребро удаляется из графа;
* С помощью поиска в глубину алгоритм пытается найти путь от одной вершины ребра к другой. Если такой путь найден, то значит, что этот путь и рассматриваемое ребро образуют простой цикл, и этот цикл добавляется в множество простых циклов графа. После этого продолжаем поиск в глубину. Если же путь не найден, то переходим к следующему ребру.

Алгоритм работает до тех пор, пока не будут удалены все ребра графа.

1. Техническое описание программного продукта

Для решения задачи была реализована функция public List<List<Vertex>> SimpleCycles(), принадлежащая классу Graph. Данная функция копирует текущий граф, и из этой копии удаляются ребра. Циклы представляются списком вершин. По окончанию работы функция возвращает список списков вершин.

1. Инструкция по эксплуатации

Для запуска третьей задачи следует нажать кнопку «Вывести все простые циклы». После нажатия кнопки появляется форма (рис. 6).

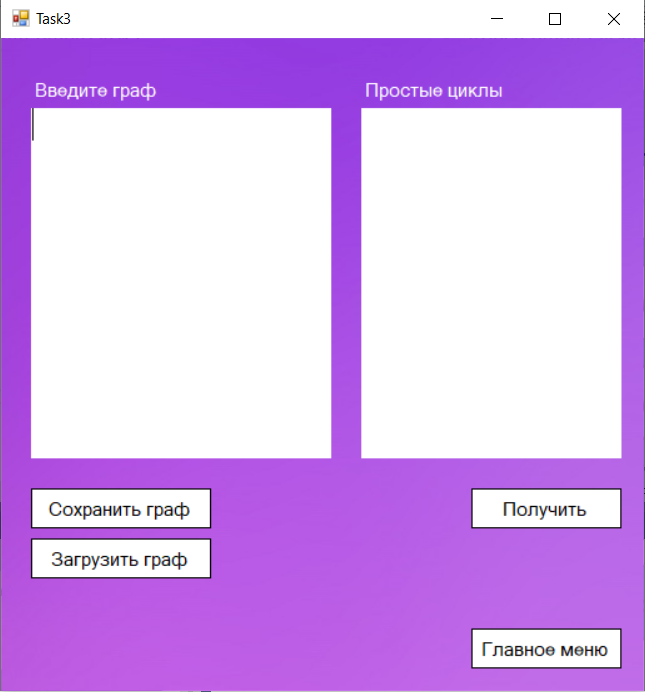


Рисунок 6. Задача 3

В поле «Простые циклы» выводится результат работы алгоритма. Введем какой-нибудь граф. Результат представлен на рисунке 7.

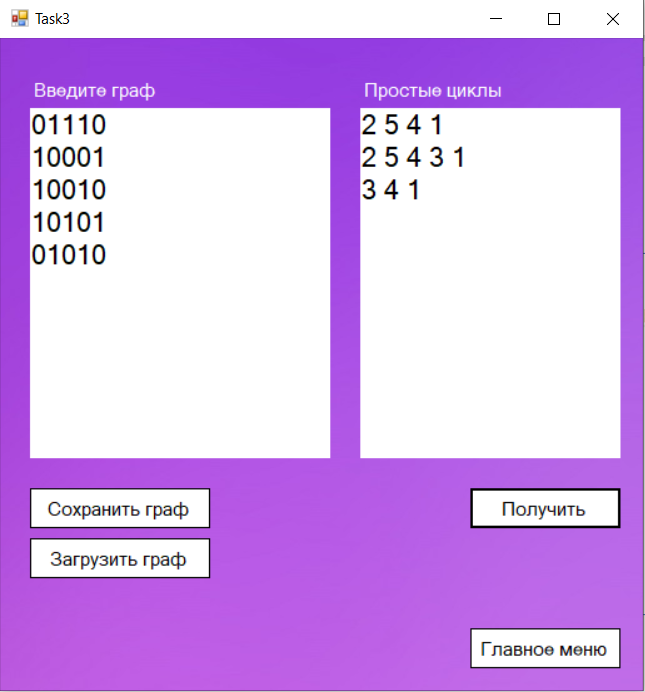
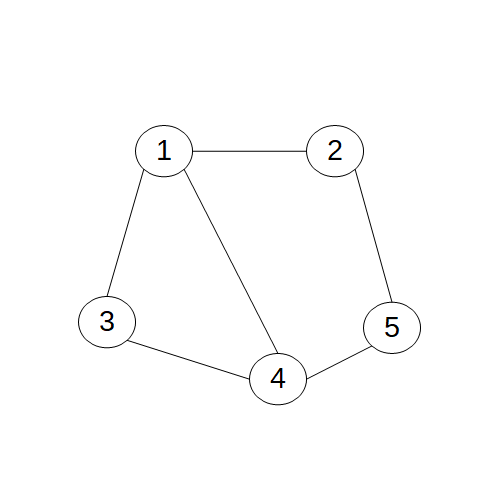
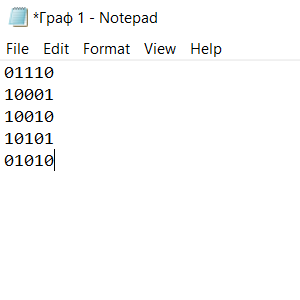


Рисунок 7. Результат задачи 3

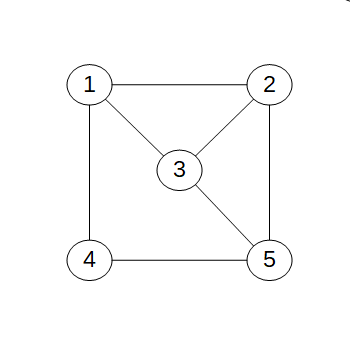
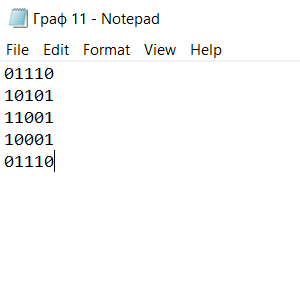
1. Набор графов для тестирования

Граф 1

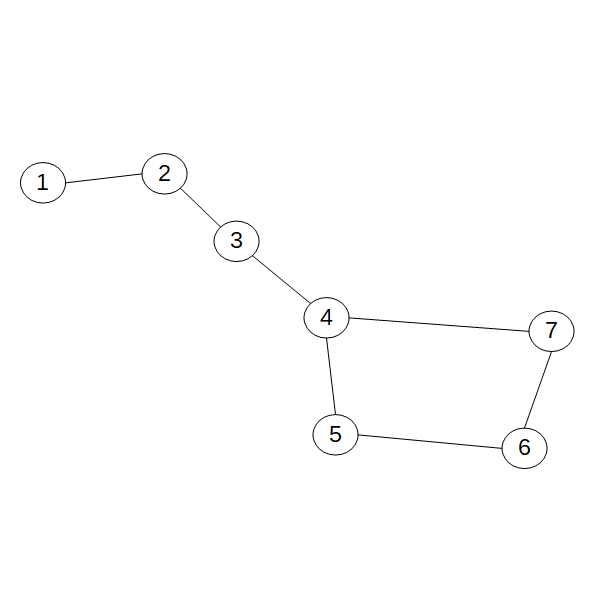
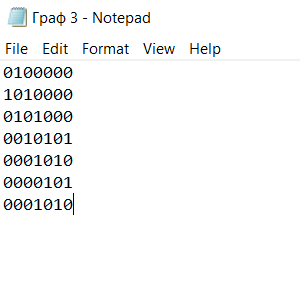
Результат: {2 5 4 1}, {2 5 4 3 1}, {3 4 1}.

Граф 2

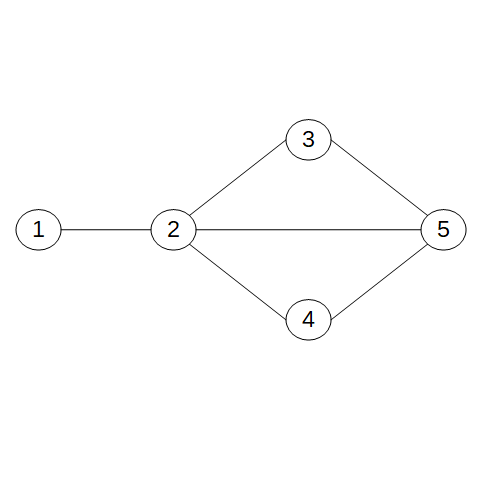
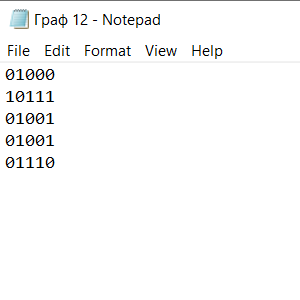
Результат: {2 3 1}, {2 3 5 4 1}, {2 5 3 1}, {2 5 4 1}, {3 2 5 4 1}, {3 5 4 1}, {3 5 2}.

Граф 3

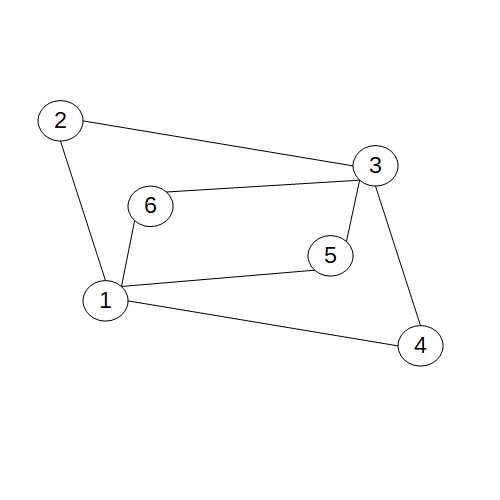
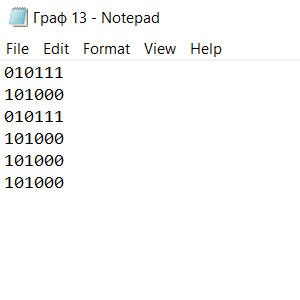
Результат: {5 6 7 4}.

Граф 4

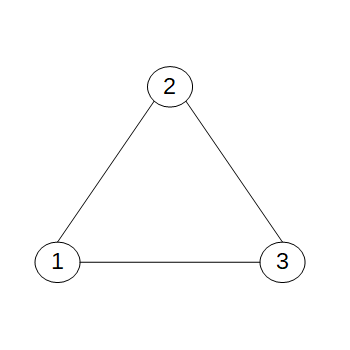
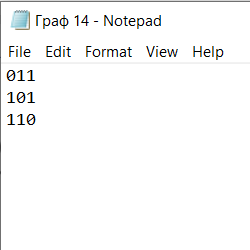
Результат: {3 5 2}, {3 5 4 2}, {4 5 2}.

Граф 5

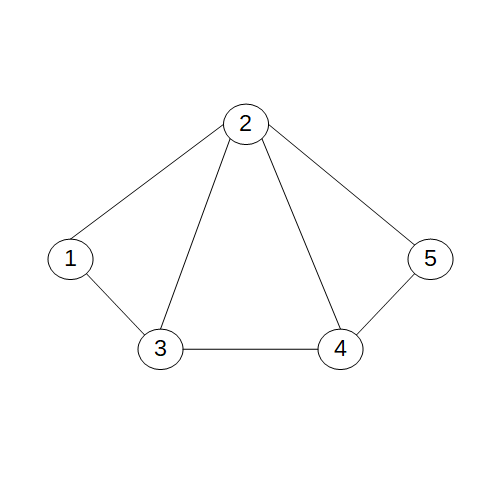
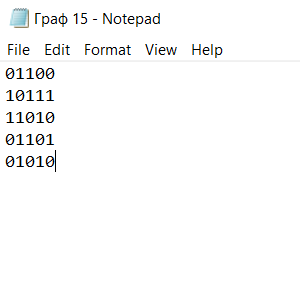
Результат: {2 3 4 1}, {2 3 5 1}, {2 3 6 1}, {4 3 5 1}, {4 3 6 1}, {5 3 6 1}.

Граф 6

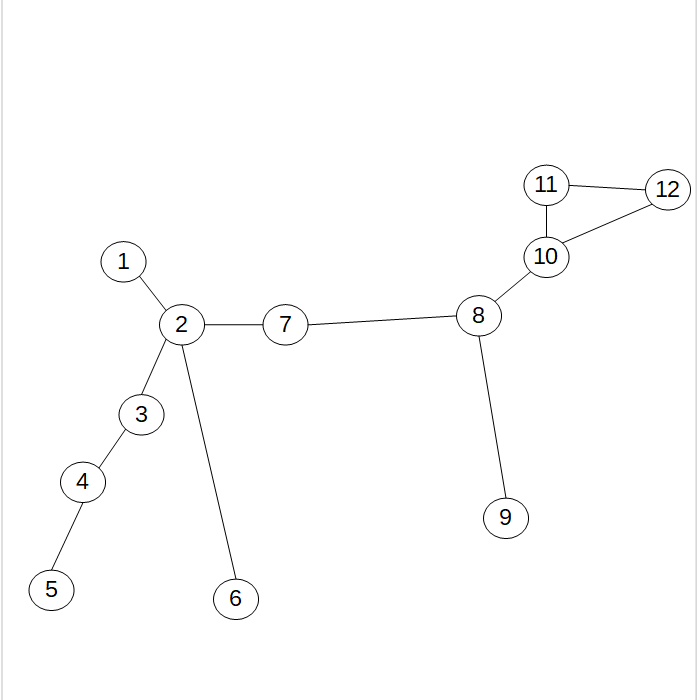
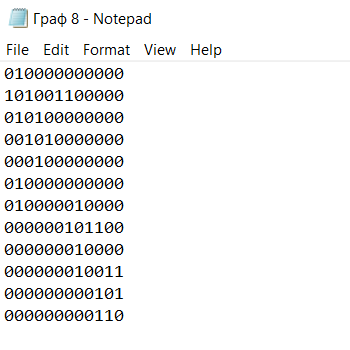
Результат: {2 3 1}.

Граф 7

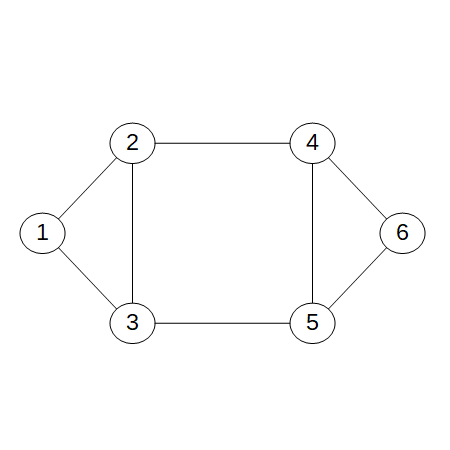
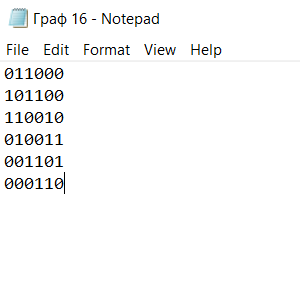
Результат: {2 3 1}, {2 4 3 1}, {2 5 4 3 1}, {3 4 2}, {3 4 5 2}, {4 5 2}.

Граф 8

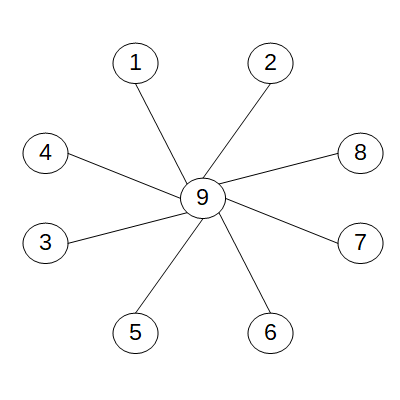
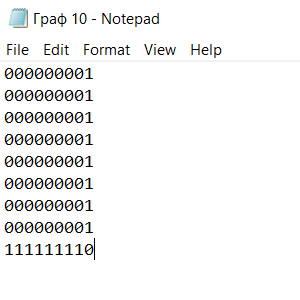
Результат: {11 12 10}.

Граф 9

Результат: {2 3 1}, {2 4 5 3 1}, {2 4 6 5 3 1}, {3 5 4 2}, {3 5 6 4 2}, {5 6 4}.

Граф 10

Результат: не найдено.

**Задание 4.**

Дан неориентированный граф. Построить наименьшее покрытие.

1. Математическая постановка задачи

Вершинное покрытие для неориентированного графа — это множество его вершин , такое, что, у каждого ребра графа хотя бы один из концов входит в вершину из *.*

Размером вершинного покрытия называется число входящих в него вершин.

Наименьшее вершинное покрытие – это покрытие наименьшего размера.

1. Описание алгоритма решения

Чтобы найти наименьшие покрытия, алгоритм просматривает все непустые подмножества множества вершин. Поиск начинается с множеств мощности 1 и далее по возрастанию. Если алгоритм находит множество, которое является вершинным покрытием, то он добавляет его в множество решений. Далее алгоритм просматриваем все оставшиеся множество той же мощности, чтобы не найти другие наименьшие вершинные покрытия, и затем останавливается.

1. Техническое описание программного продукта

Для решения задачи была реализована функция public SmallestVertexCoverage(), принадлежащая классу Graph. Данная функция просматривается все непустые подмножества вершин и в качестве ответа возвращает список наименьших вершинных покрытий (список списков вершин).

1. Инструкция по эксплуатации

Данной задаче соответствует кнопка «Построить наименьшее вершинное покрытие». Ввод данных производится в форме, представленной на рисунке 8.

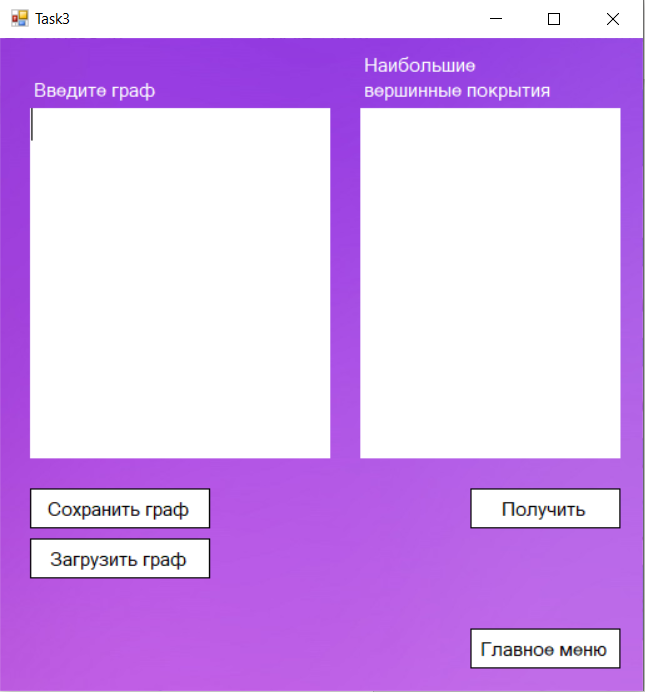


Рисунок 8. Задача 3

Результат выводится в поле «Наибольшие вершинные покрытия». Проверим работу алгоритма. Введем граф и нажмем кнопку «Получить». Программа выводит наибольшие вершинные покрытия (рис. 9).

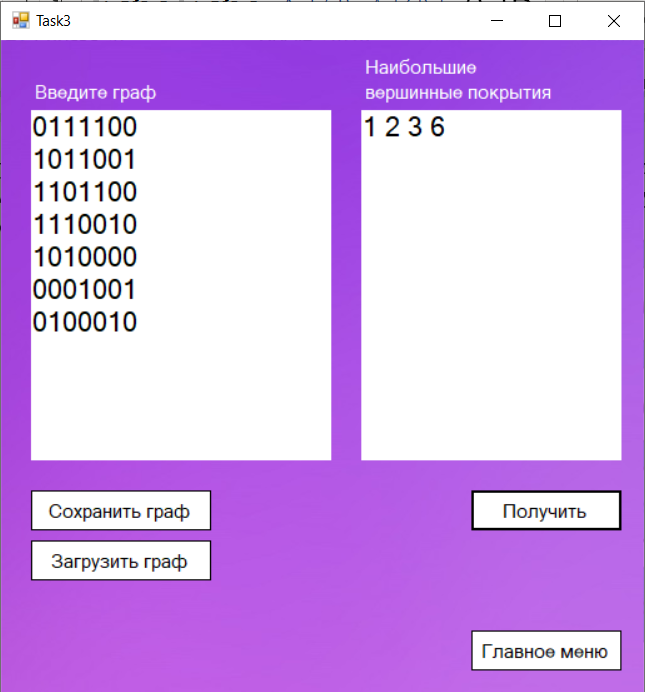
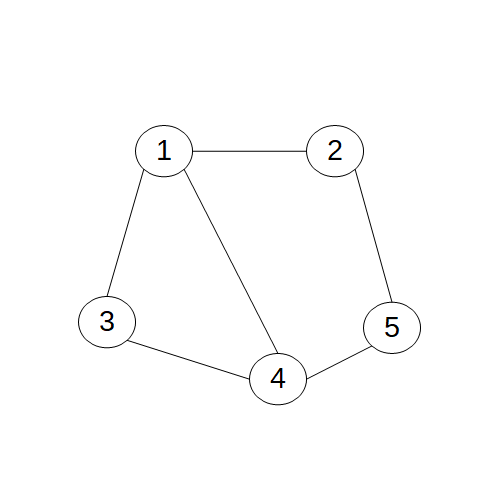
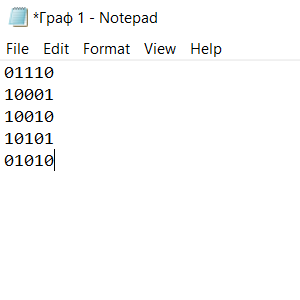


Рисунок 9. Результат работы задачи 4

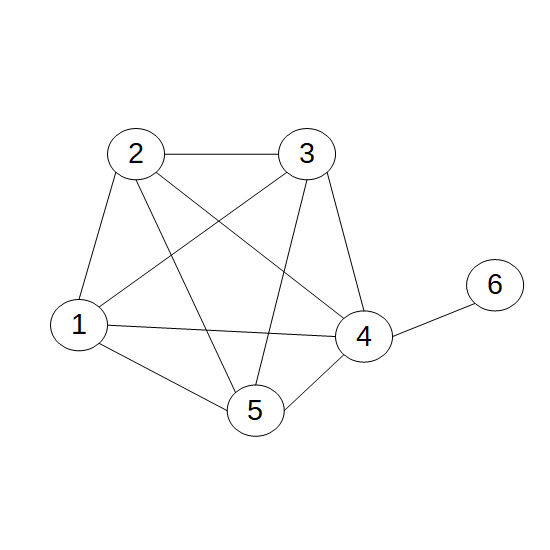
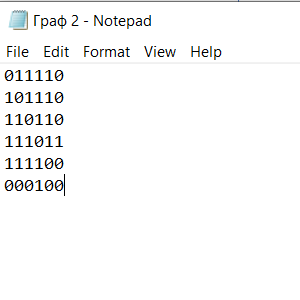
1. Набор графов для тестирования

Граф 1

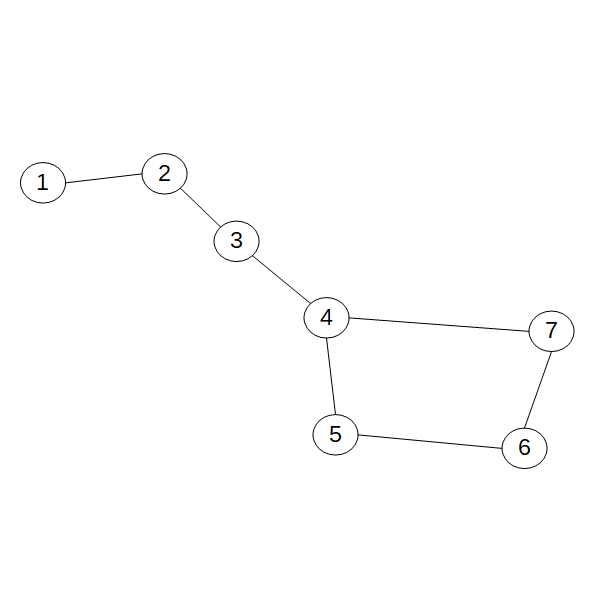
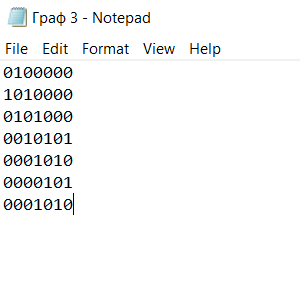
Результат: {1 2 4}.

Граф 2

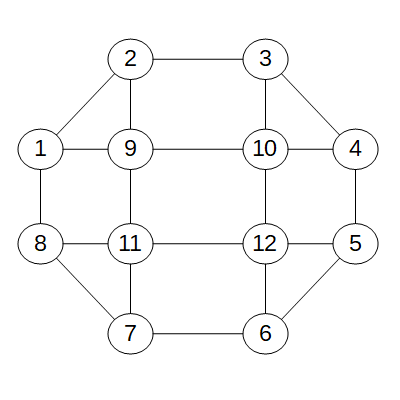
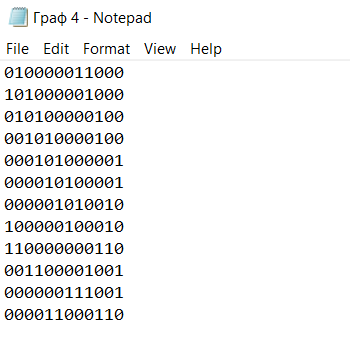
Результат: {1 2 3 4}.

Граф 3

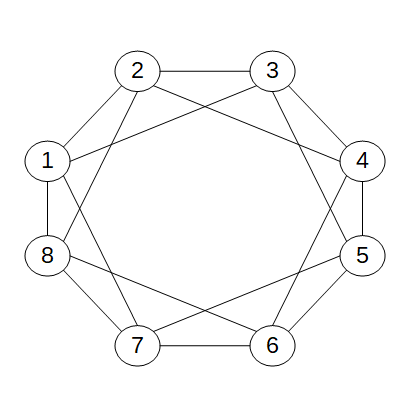
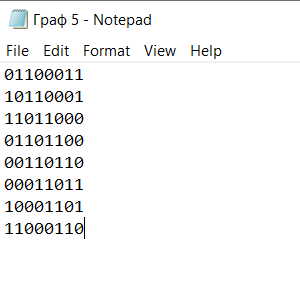
Результат: { 1 2 3 4 6}.

Граф 4

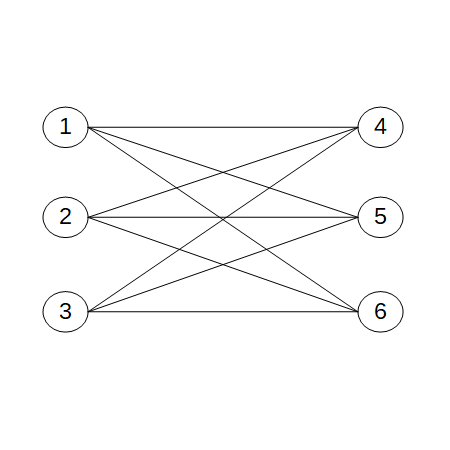
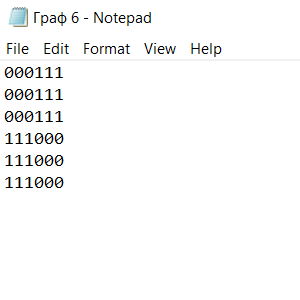
Результат: { 1 2 3 4 5 6 7 8 9 12}.

Граф 5

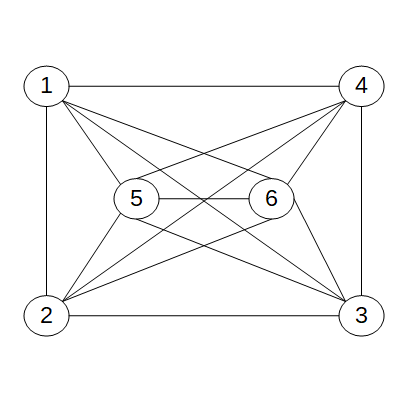
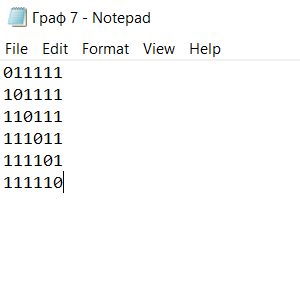
Результат: {1 2 3 4 5 6 7}, {1 2 3 4 5 6 8}.

Граф 6

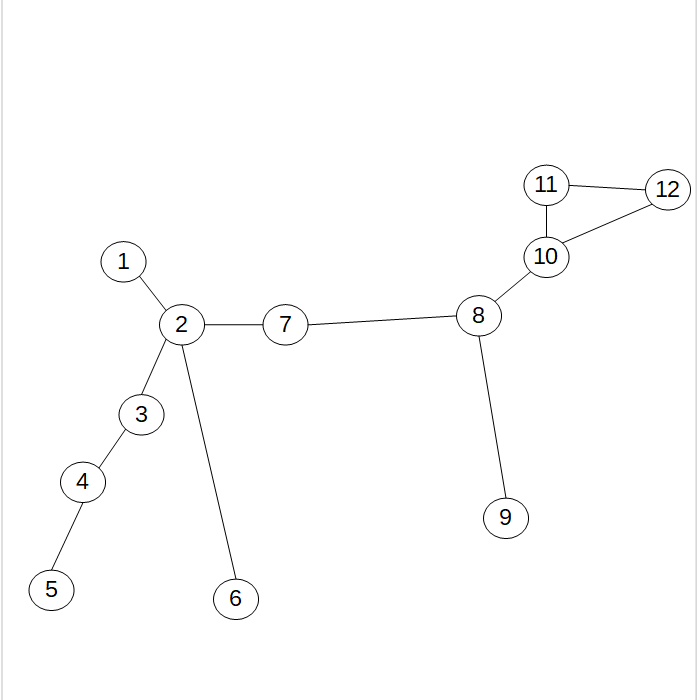
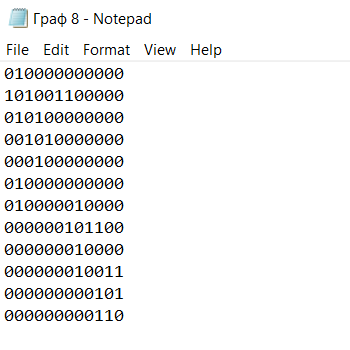
Результат: {1 2 3}.

Граф 7

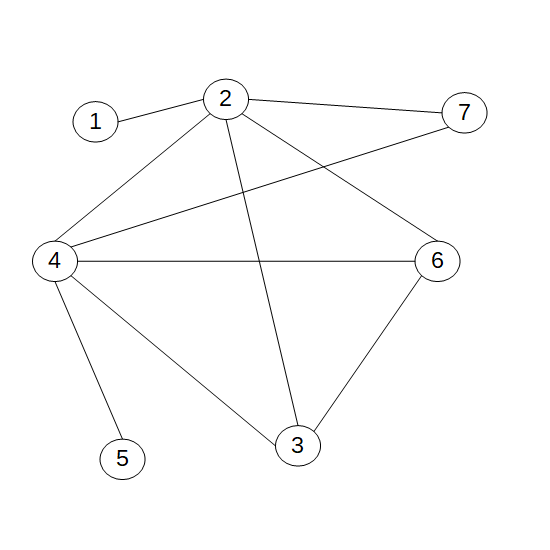
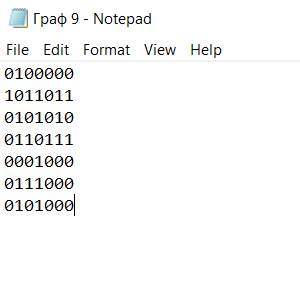
Результат: {1 2 3 4 5}, {1 2 3 4 6}.

Граф 8

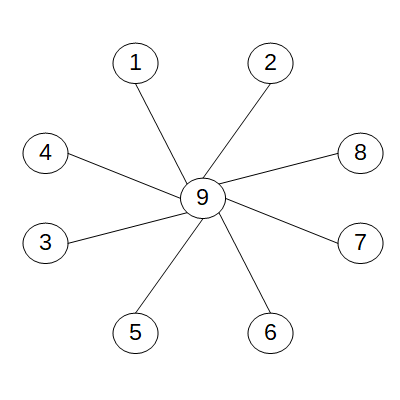
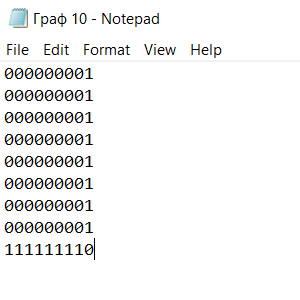
Результат: {1 2 3 4 5 6 7 8 9 10 11}, {1 2 3 4 5 6 7 8 9 10 12}.

Граф 9

Результат: {1 2 3 4}.

Граф 10

Результат: {9}.

**Задание 5.**

Реализовать алгоритм Ли нахождения кратчайшего пути. Описать возможности его применения.

1. Математическая постановка задачи

Алгори́тм волново́й трассиро́вки (волновой алгоритм, алгоритм Ли) — алгоритм поиска пути, алгоритм поиска кратчайшего пути на планарном графе. Принадлежит к алгоритмам, основанным на методах поиска в ширину.

Волновой алгоритм — один из основных при автоматизированной трассировке (разводке) печатных плат. Также одно из характерных применений волнового алгоритма — поиск кратчайшего расстояния на карте в стратегических играх.

1. Описание алгоритма решения

Обозначим – начальная вершина пути, – конечная.

Алгоритм помечает вершину как нулевую. Остальные вершины считаются непомеченными.

На -той итерации помечаются индексом все непомеченные ранее вершины, в которых есть дуги от помеченных вершин. Если в текущем значении индекса оказались помеченными какие-либо вершины, то . Иначе делается вывод, что пути из вершины в нет.

Если среди помеченных вершин есть вершина , то обратным проходом по дугам начиная от вершины выделяются те дуги, которые инцидентны выделенным вершинам и разность между весами которых равна 1. При движении от вершины по выделенным дугам оказывается построены все кратчайшие пути к вершине . Алгоритм завершен.

1. Техническое описание программного продукта

Для решения задачи была реализована функция public SmallestVertexCoverage(), принадлежащая классу Graph. Данная функция просматривается все непустые подмножества вершин и в качестве ответа возвращает список наименьших вершинных покрытий (список списков вершин).

1. Инструкция по эксплуатации

Нажмем на кнопку «Алгоритм Ли нахождения кратчайшего пути». Появится следующая форма (рис. 10).

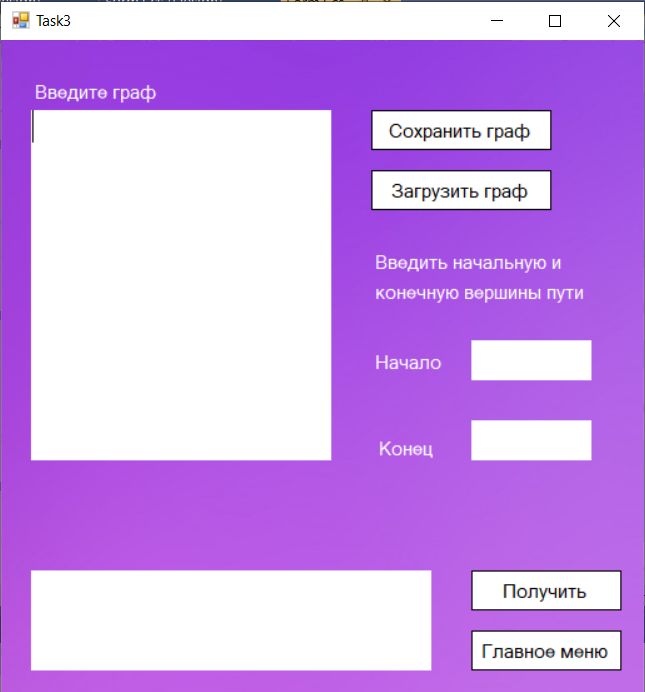


Рисунок 10. Задача 5

Результат выводится в поле «Наибольшие вершинные покрытия». Проверим работу алгоритма. Введем граф и нажмем кнопку «Получить». Программа выводит наибольшие вершинные покрытия (рис. 9).

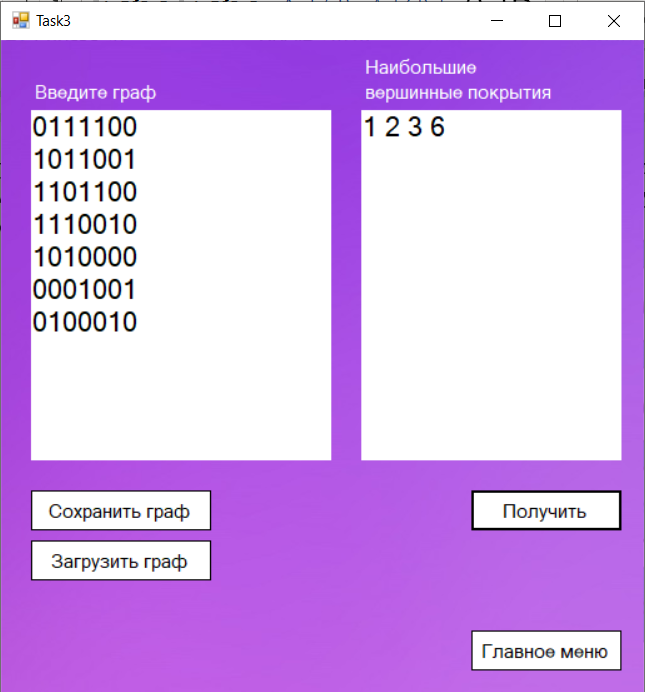
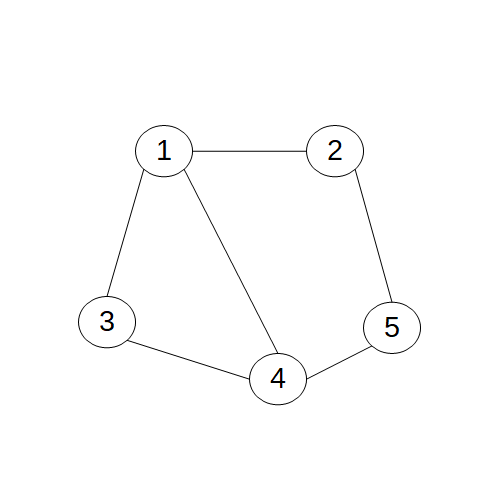
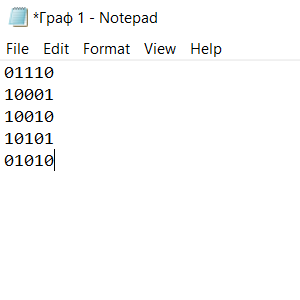


Рисунок 12. Результат работы задачи 4

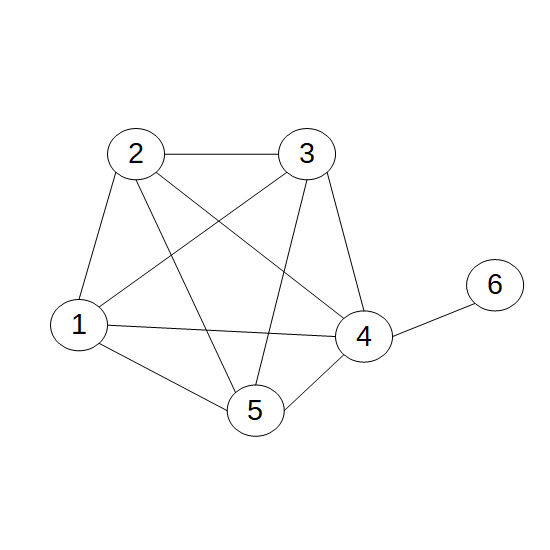
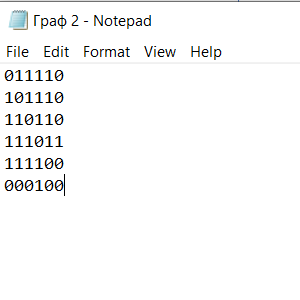
1. Набор графов для тестирования

Граф 1

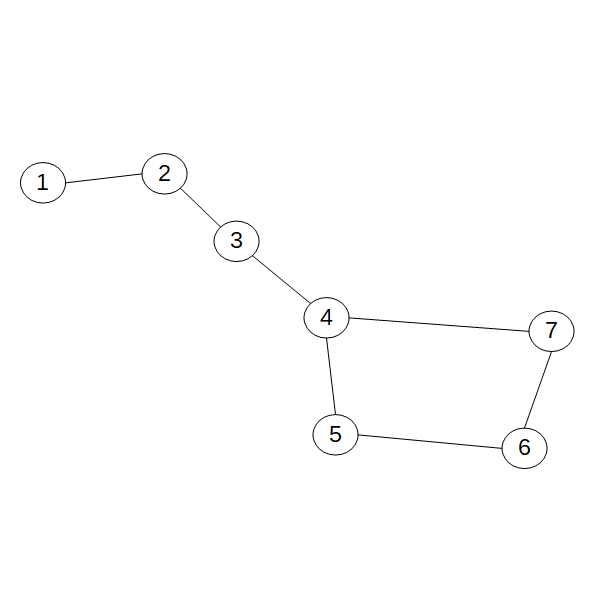
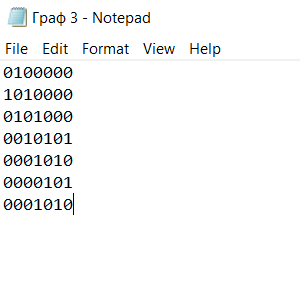
Начало: 2. Конец: 5. Результат: 2-5.

Граф 2

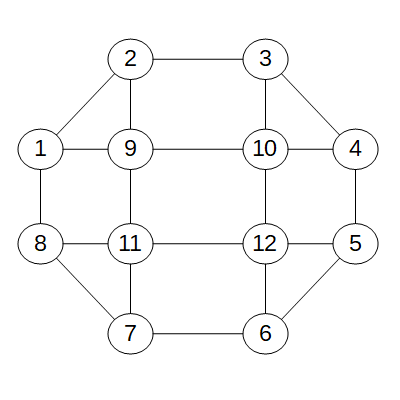
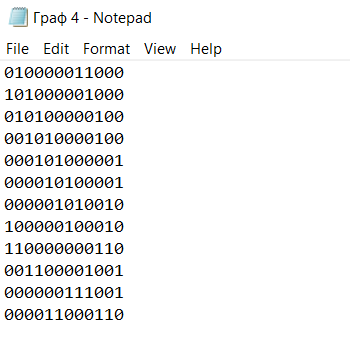
Начало: 1. Конец: 6. Результат: 1-4-6.

Граф 3

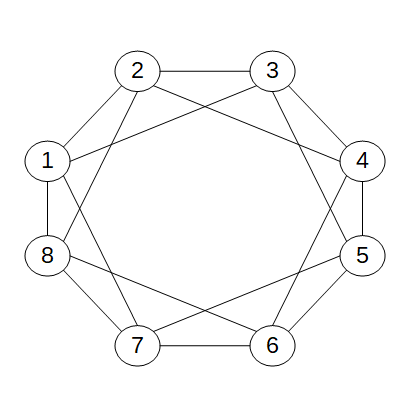
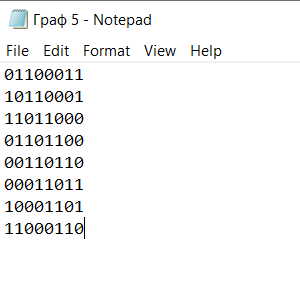
Начало: 1. Конец: 6. Результат: 1-2-3-4-5-6, 1-2-3-4-7-6.

Граф 4

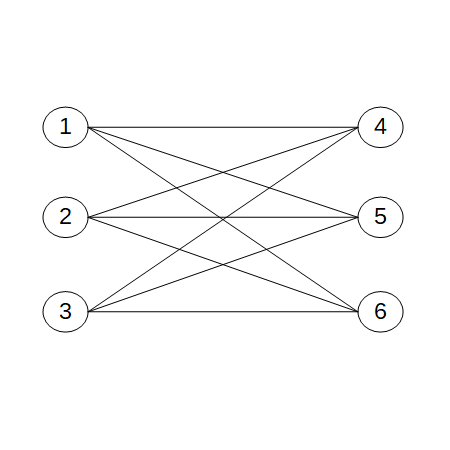
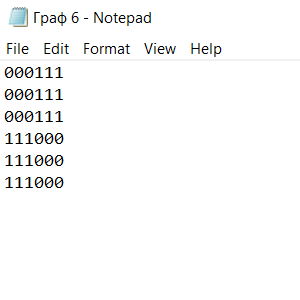
Начало: 2. Конец: 12. Результат: 2-3-10-12, 2-9-10-12, 2-9-11-12.

Граф 5

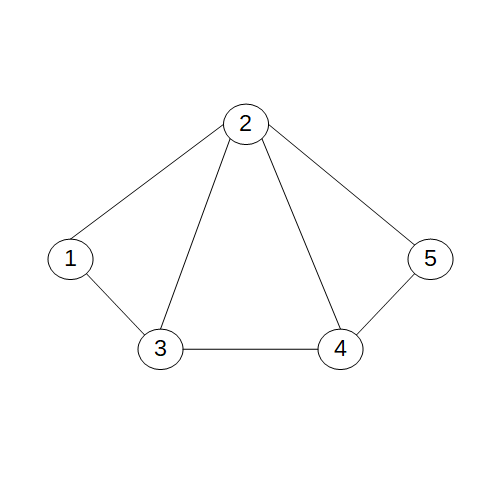
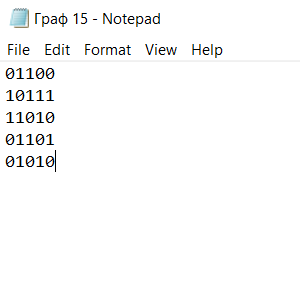
Начало: 3. Конец: 6. Результат: 3-4-6, 3-5-6.

Граф 6

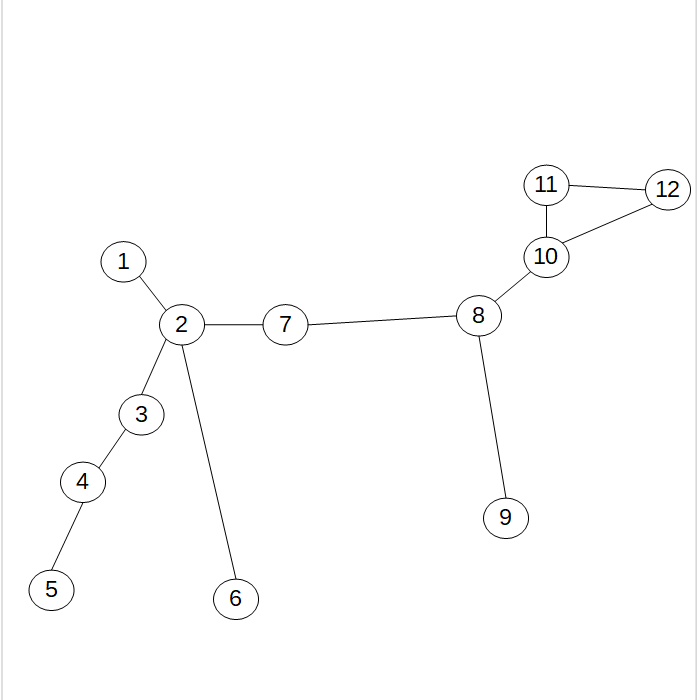
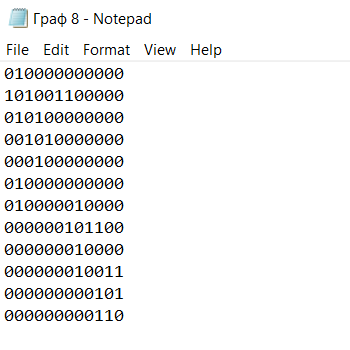
Начало: 4. Конец: 5. Результат: 4-1-5, 4-2-5, 4-3-5.

Граф 7

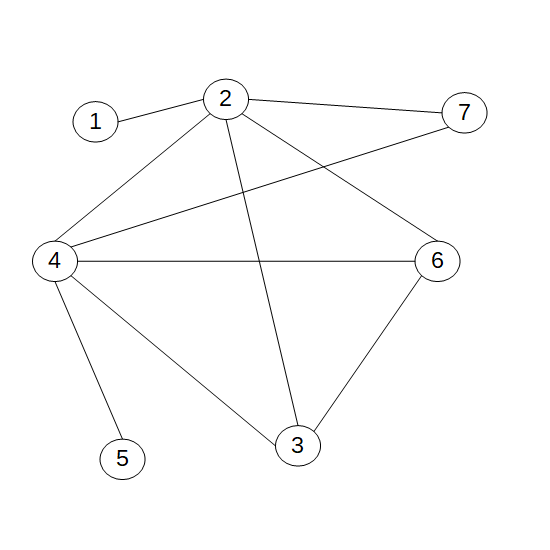
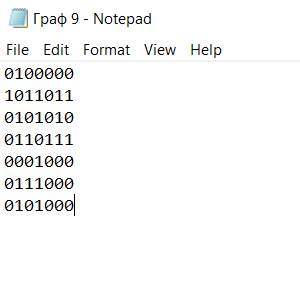
Начало: 1. Конец: 4. Результат: 1-2-4, 1-3-4.

Граф 8

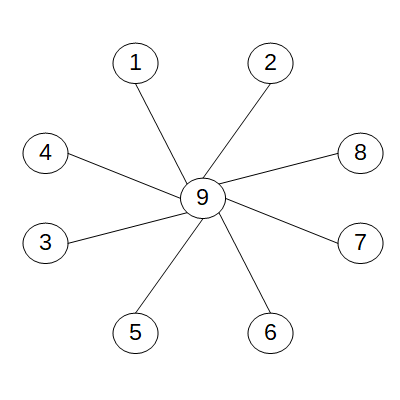
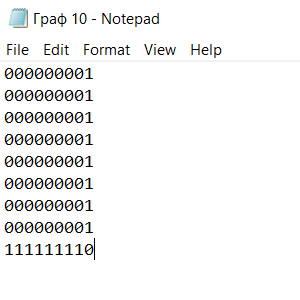
Начало: 9. Конец: 12. Результат: 9-8-10-12.

Граф 9

Начало: 7. Конец: 6. Результат: 7-2-6, 7-4-6.

Граф 10

Начало: 9. Конец: 1. Результат: 9-1.