МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ОТЧЁТ**

**ПО ИНДИВИДУАЛЬНОЙ РАБОТЕ №1**

**ДИСЦИПЛИНА : «КОНСТРУИРОВАНИЕ АЛГОРИМОВ И СТРУКТУР ДАННЫХ»**

Работу выполнил\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Хван К.Л.

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Компьютерные науки

Преподаватель

ассистент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Климец

Краснодар

2020

**Первая задача**

1. Математическая постановка задачи

Пусть множество  - это математический объект, являющийся набором, собранием каких-либо объектов, которые называются элементами этого множества и обладают общим для всех их характеристическим свойством. Подмножеством  назовем множество, элементы которого входят в другое множество, по отношению к которому данное множество и является подмножеством.

Прямым или декартовым произведением двух множеств назовем множество  элементами которого являются все возможные упорядоченные пары элементов исходных множеств.

Под графом будем понимать математический объект, обозначаемый как, где ,  - множество вершин графа, множество (в случае неориентированного графа -множество неупорядоченных пар вершин) ребер графа, являющееся подмножеством декартового произведения множества вершин на само себя.

Граф назовём связным, если для любых двух вершин существует путь из одной в другую.

Под путём будем понимать последовательность вершин  в которой каждая вершина соединена со следующей вершиной ребром.

Эйлеровым путем в графе называется путь, который проходит по каждому ребру, причем ровно один раз.

Эйлеров цикл - замкнутый эйлеров путь.

Граф называется эйлеровым, если он содержит эйлеров цикл. Граф называется полуэйлеровым, если он содержит эйлеров путь, но не содержит эйлеров цикл.

Для того, чтобы граф был эйлеровым необходимо чтобы все вершины имели четную степень.

Чтобы проверить, существует ли эйлеров путь, нужно воспользоваться следующей теоремой. Эйлеров цикл существует тогда и только тогда, когда степени всех вершин чётны. Эйлеров путь существует тогда и только тогда, когда количество вершин с нечётными степенями равно двум (или нулю, в случае существования эйлерова цикла).

1. Описание алгоритма решения.

Сначала проверим, существует ли эйлеров путь. Затем найдём все простые циклы и объединим их в один - это и будет эйлеровым циклом. Если граф таков, что эйлеров путь не является циклом, то, добавим недостающее ребро, найдём эйлеров цикл, потом удалим лишнее ребро.

Алгоритм на псевдокоде:

stack St;

в St кладём любую вершину (стартовая вершина);

пока St не пустой

пусть V - значение на вершине St;

если степень(V) = 0, то

добавляем V к ответу;

снимаем V с вершины St;

иначе

находим любое ребро, выходящее из V;

удаляем его из графа;

второй конец этого ребра кладём в St;

Сначала программа проверяет степени вершин: если вершин с нечётной степенью нет, то в графе есть эйлеров цикл, если есть 2 вершины с нечётной степенью, то в графе есть только эйлеров путь (эйлерова цикла нет), если же таких вершин больше 2, то в графе нет ни эйлерова цикла, ни эйлерова пути. Чтобы найти эйлеров путь (не цикл), поступим таким образом: если V1 и V2 - это две вершины нечётной степени, то просто добавим ребро (V1,V2), в полученном графе найдём эйлеров цикл (он, очевидно, будет существовать), а затем удалим из ответа "фиктивное" ребро (V1,V2). Эйлеров цикл будем искать в точности так, как описано выше, и заодно по окончании этого алгоритма проверим, связный был граф или нет (если граф был не связный, то по окончании работы алгоритма в графе останутся некоторые рёбра, и в этом случае нам надо вывести -1). Наконец, программа учитывает, что в графе могут быть изолированные вершины.

1. Техническое описание программного продукта

Алгоритм реализован на языке программирования C#. Применены классы «Algorithms.cs», «EdgeGraph», «VertexGraph», «Graph», «Form1.cs», «Form1.Designer.cs»,«Programm.cs».

Были использованы:

* Объект класса «Graph» graph, содержащий в себе список вершин, представимых объектами класса «VertexGraph» и список ребер, представимых объектами класса «EdgeGraph».
* Конструкторы класса «Graph»:
* public Graph() – создает пустые список вершин и список ребер.
* public Graph(string InputText) – создает и инициализирует список вершин и список ребер.
* Методы класса «Graph»:
* public void AddVertex(VertexGraph vertex) – добавляет вершину в список вершин.
* public void AddEdge(VertexGraph from, VertexGraph to) – добавляет ребро в список ребер с весом 1.
* public void AddEdge(VertexGraph from, VertexGraph to, int weight) – добавляет ребро в список ребер с заданным весом.
* public int [,] GetMatrix() – строит матрицу смежности и возвращает её.
* public List<VertexGraph> GetVertices() – создает копию списка вершин графа и возвращает её.
* public List<EdgeGraph> GetEdges() - создает копию списка ребер графа и возвращает её.
* public List<VertexGraph> GetVertexLists(VertexGraph vertex) – возвращает список вершин смежных с вершиной vertex.
* Свойства класса «Graph»:
* public int VertexCount – возвращает кол-во вершин в списке вершин.
* public int EdgesCount – возвращает кол-во ребер в списке ребер.
* Конструктор класса «VertexGraph»:
* public VertexgGraph(int number) – задает номер вершины.
* Свойства класса «VertexGraph»:
* public int Number – возвращает или задает номер вершины.
* Методы класса «VertexGraph»:
* public override string ToString() – переводит номер вершины из типа int в тип string.
* Конструктор класса «EdgesGraph»:
* public EdgeGraph(VertexGraph from, VertexGraph to,int weight = 1) – задает ребро в виде «откуда - куда - вес ребра», (если не задан параметр weight, то по умолчанию вес ребра равен 1)
* Свойства класса «EdgesGraph»:
* public VertexGraph From { get; set; } – задает или возвращает вершину вида «откуда».
* public VertexGraph To { get; set; } – задает или возвращает вершину вида «куда».
* public int Weight { get; set; } - задает или возвращает вес ребра.
* Методы класса «EdgesGraph»:
* public override string ToString() – возвращает строку вида   
  (a; b; weight)
* Методы класса «Algorithm»:
* public bool FindEulerPath(Graph graph, out List<int>R) – возвращает true - если есть эйлеров путь, false – иначе. Также возвращает список вершин R – сам эйлеров путь.

1. Инструкция по эксплуатации.

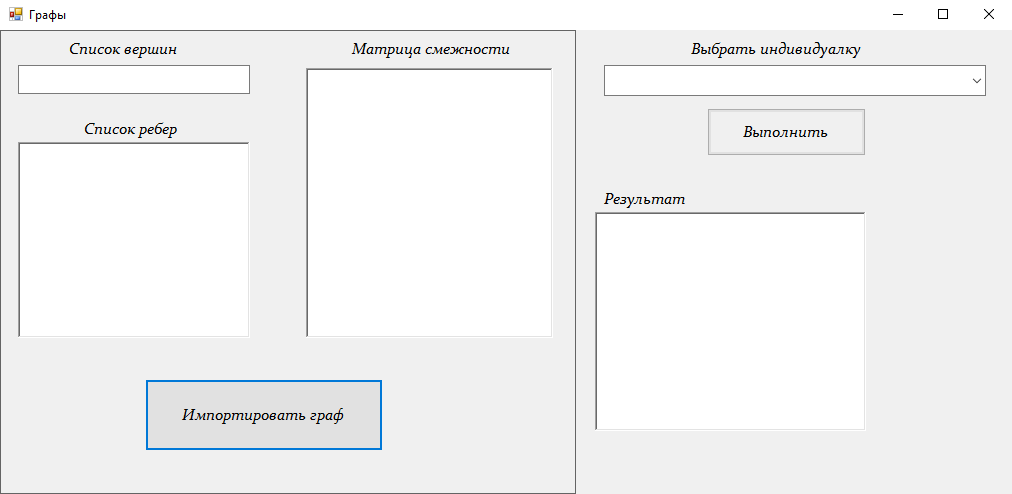
При запуске программы пользователь должен нажать кнопку «Импортировать граф» и выбрать текстовый файл с расширением .txt, где находится граф в виде таблицы смежности.

Рисунок 1 – Исходное состояние окна программы

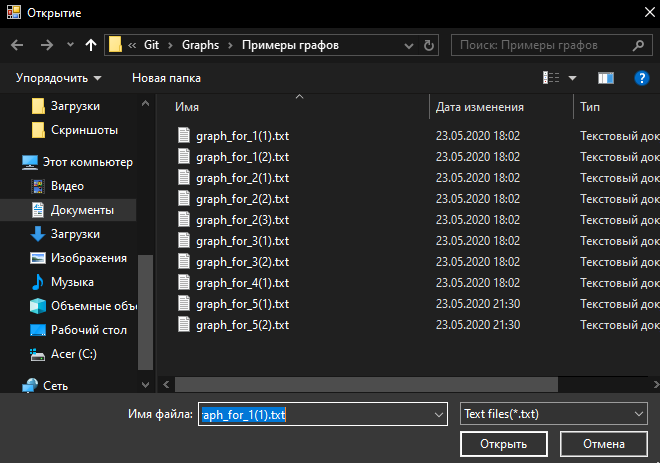
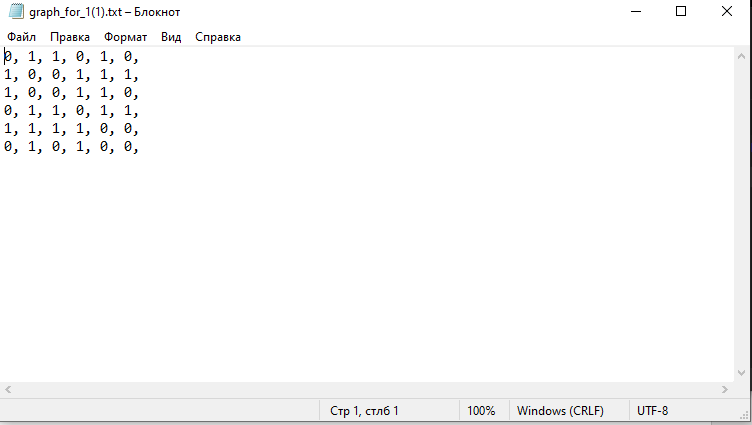


Рисунок 3 – представление графа

Рисунок 2 – Состояние окна программы после нажатия кнопки «Импортировать граф»

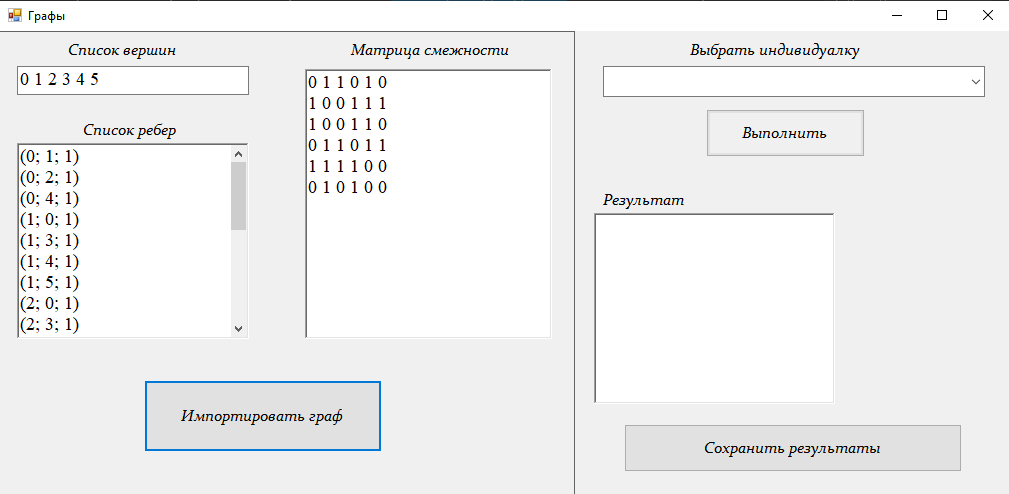


Рисунок 4 – Состояние окна программы после выбора графа

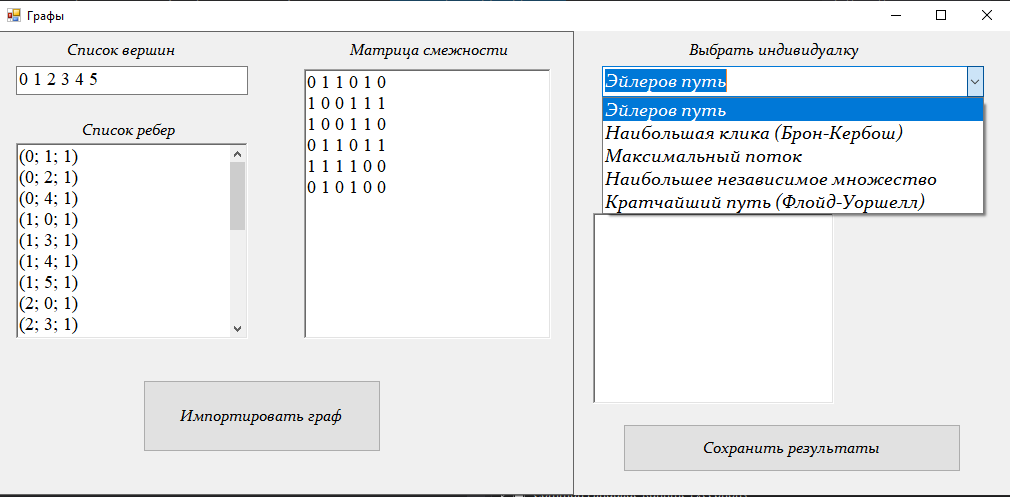
Затем пользователю требуется выбрать из выпадающего списка нужную задачу.

Рисунок 5 – Состояние окна программы при выборе задачи

Далее пользователь должен нажать на кнопку «Выполнить».

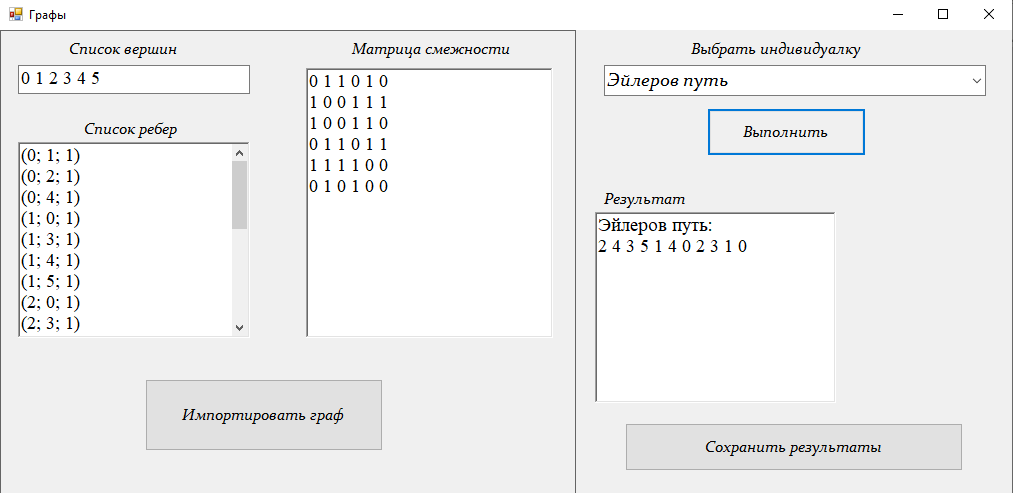
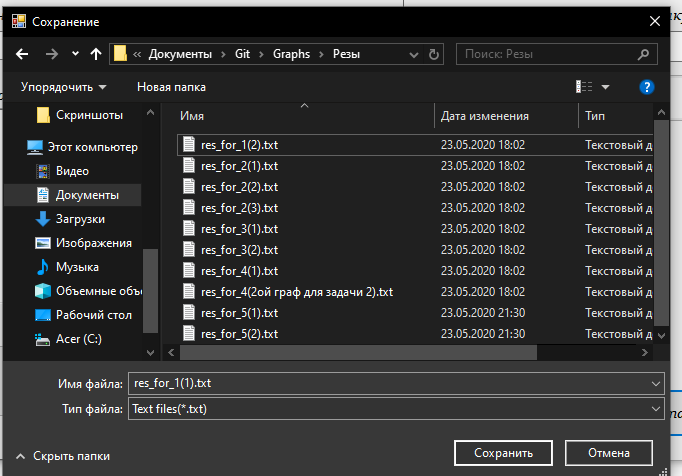
Если пользователь захочет сохранить результат, то он должен нажать на кнопку «Сохранить результаты», выбрать имя сохраняемого файла и путь, куда файл будет записан.

Рисунок 5 – Состояние окна программы после нажатия кнопки «Выполнить»

Рисунок 5 – Состояние окна программы после нажатия кнопки

«Сохранить результаты»

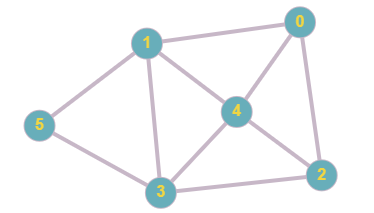
1. Набор графов для тестирования.

Рисунок 6 – Первый граф для тестирования

Матрица смежности графа:

0, 1, 1, 0, 1, 0,

1, 0, 0, 1, 1, 1,

1, 0, 0, 1, 1, 0,

0, 1, 1, 0, 1, 1,

1, 1, 1, 1, 0, 0,

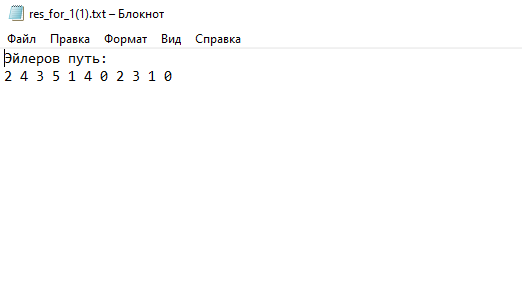
0, 1, 0, 1, 0, 0,

Рисунок 7 – Результат работы программы на первом графе

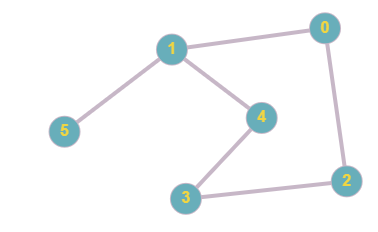


Рисунок 8 – Второй граф для тестирования

Матрица смежности графа:

0, 1, 1, 0, 1, 0,

1, 0, 0, 1, 1, 1,

1, 0, 0, 1, 1, 0,

0, 1, 1, 0, 1, 1,

1, 1, 1, 1, 0, 0,

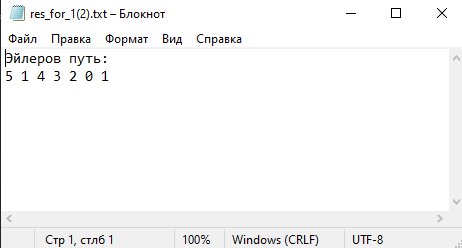
0, 1, 0, 1, 0, 0,

Рисунок 9 – Результат работы программы на втором графе

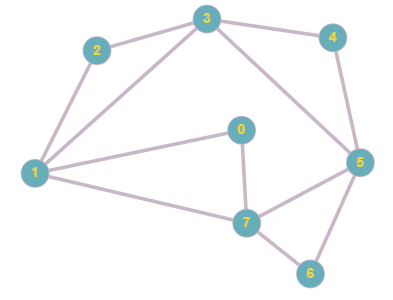
Матрица смежности графа:

Рисунок 10 – Третий граф для тестирования

0, 1, 0, 0, 0, 0, 0, 1,

1, 0, 1, 1, 0, 0, 0, 1,

0, 1, 0, 1, 0, 0, 0, 0,

0, 1, 1, 0, 1, 1, 0, 0,

0, 0, 0, 1, 0, 1, 0, 0,

0, 0, 0, 1, 1, 0, 1, 1,

0, 0, 0, 0, 0, 1, 0, 1,

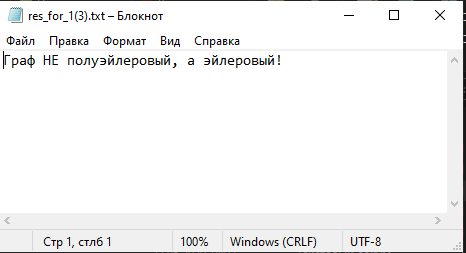
1, 1, 0, 0, 0, 1, 1, 0,

Рисунок 11 – Результат работы программы на третьем графе

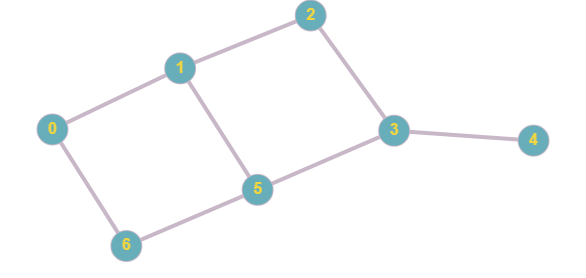
****

Рисунок 12 – Четвертый граф для тестирования

Матрица смежности графа:

0, 1, 0, 0, 0, 0, 1,

1, 0, 1, 0, 0, 1, 0,

0, 1, 0, 1, 0, 0, 0,

0, 0, 1, 0, 1, 1, 0,

0, 0, 0, 1, 0, 0, 0,

0, 1, 0, 1, 0, 0, 1,

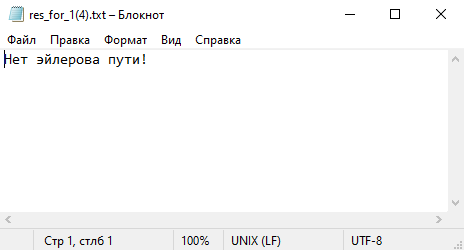
1, 0, 0, 0, 0, 1, 0,

Рисунок 13 – Результат работы программы на четвертом графе

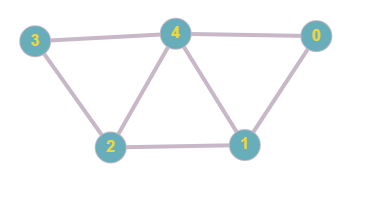


Рисунок 14 – Пятый граф для тестирования

Матрица смежности графа:

0, 1, 0, 0, 1,

1, 0, 1, 0, 1,

0, 1, 0, 1, 1,

0, 0, 1, 0, 1,

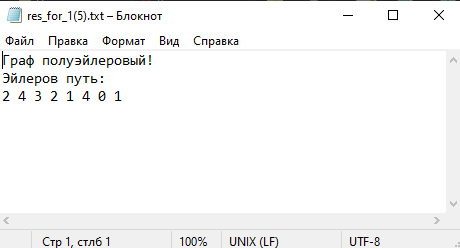
1, 1, 1, 1, 0,

Рисунок 15 – Результат работы программы на пятом графе

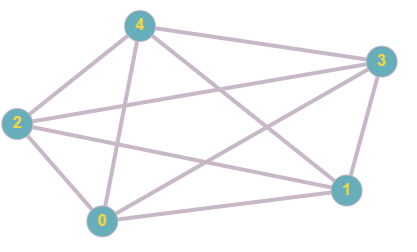


Рисунок 16 – Шестой граф для тестирования

Матрица смежности графа:

0, 1, 1, 1, 1,

1, 0, 1, 1, 1,

1, 1, 0, 1, 1,

1, 1, 1, 0, 1,

1, 1, 1, 1, 0,

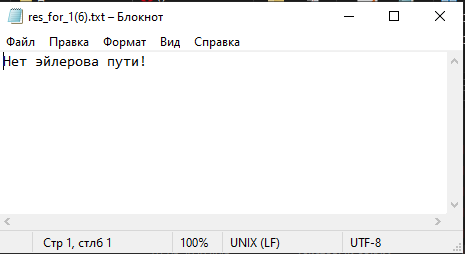


Рисунок 17 – Результат работы программы на шестом графе

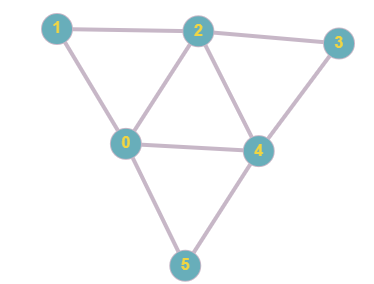


Рисунок 18 – Седьмой граф для тестирования

Матрица смежности графа:

0, 1, 1, 0, 1, 1,

1, 0, 1, 0, 0, 0,

1, 1, 0, 1, 1, 0,

0, 0, 1, 0, 1, 0,

1, 0, 1, 1, 0, 1,

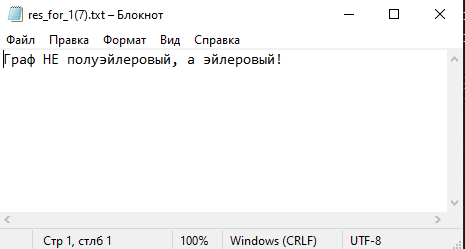
1, 0, 0, 0, 1, 0,

Рисунок 19 – Результат работы программы на седьмом графе

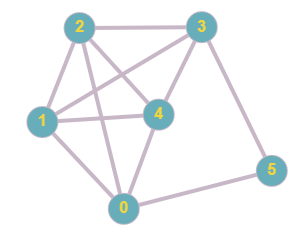


Рисунок 20 – Восьмой граф для тестирования

Матрица смежности графа:

0, 1, 1, 0, 1, 1,

1, 0, 1, 1, 1, 0,

1, 1, 0, 1, 1, 0,

0, 1, 1, 0, 1, 1,

1, 1, 1, 1, 0, 0,

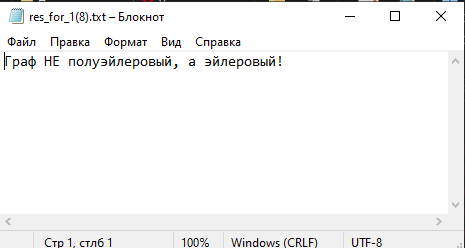
1, 0, 0, 1, 0, 0,

Рисунок 21 – Результат работы программы на восьмом графе

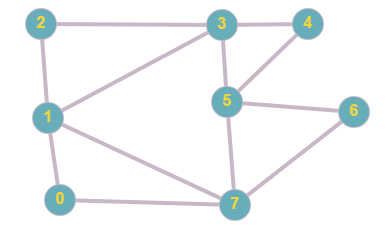


Рисунок 22 – Девятый граф для тестирования

Матрица смежности графа:

0, 1, 0, 0, 0, 0, 0, 1,

1, 0, 1, 1, 0, 0, 0, 1,

0, 1, 0, 1, 0, 0, 0, 0,

0, 1, 1, 0, 1, 1, 0, 0,

0, 0, 0, 1, 0, 1, 0, 0,

0, 0, 0, 1, 1, 0, 1, 1,

0, 0, 0, 0, 0, 1, 0, 1,

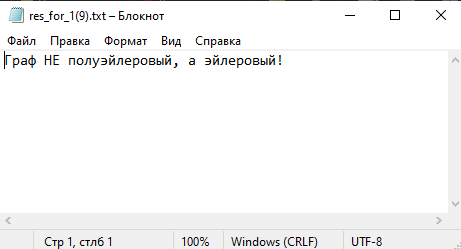
1, 1, 0, 0, 0, 1, 1, 0,

Рисунок 21 – Результат работы программы на девятом графе

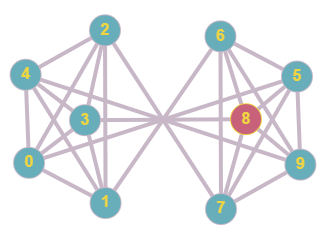


Рисунок 22 – Десятый граф для тестирования

Матрица смежности графа:

0, 1, 1, 1, 1, 1, 0, 0, 0, 0,

1, 0, 1, 1, 1, 0, 1, 0, 0, 0,

1, 1, 0, 1, 1, 0, 0, 1, 0, 0,

1, 1, 1, 0, 1, 0, 0, 0, 1, 0,

1, 1, 1, 1, 0, 0, 0, 0, 0, 1,

1, 0, 0, 0, 0, 0, 1, 1, 1, 1,

0, 1, 0, 0, 0, 1, 0, 1, 1, 1,

0, 0, 1, 0, 0, 1, 1, 0, 1, 1,

0, 0, 0, 1, 0, 1, 1, 1, 0, 1,

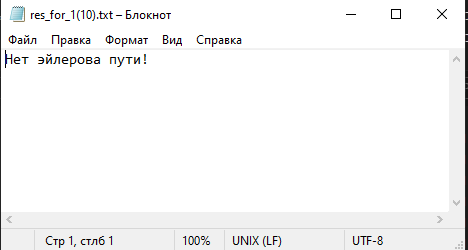
0, 0, 0, 0, 1, 1, 1, 1, 1, 0,

Рисунок 23 – Результат работы программы на десятом графе

**Вторая задача**

1. Математическая постановка задачи

Полный граф(клика) - граф, в котором каждая пара различных вершин смежна. Полный граф с  вершинами имеет рёбер  и обозначается .

Дано: связный неориентированный граф без петель ,. Требуется построить наибольшую клику.

1. Описание алгоритма решения

Алгоритм использует тот факт, что всякая клика в графе является его максимальным по включению полным подграфом. Начиная с одиночной вершины (образующей полный подграф), алгоритм на каждом шаге пытается увеличить уже построенный полный подграф, добавляя в него вершины из множества кандидатов. Высокая скорость обеспечивается отсечением при переборе вариантов, которые заведомо не приведут к построению клики, для чего используется дополнительное множество, в которое помещаются вершины, которые уже были использованы для увеличения полного подграфа.

Алгоритм оперирует тремя множествами вершин графа:

Множество compsub — множество, содержащее на каждом шаге рекурсии полный подграф для данного шага. Строится рекурсивно.

Множество candidates — множество вершин, которые могут увеличить compsub

Множество not — множество вершин, которые уже использовались для расширения compsub на предыдущих шагах алгоритма.

Алгоритм является [рекурсивной](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F) процедурой, применяемой к этим трем множествам.

**ПРОЦЕДУРА** *extend* (*candidates*, *not*):

**ПОКА** *candidates* НЕ пусто **И** *not* НЕ содержит вершины, СОЕДИНЕННОЙ СО ВСЕМИ вершинами из *candidates*,

**ВЫПОЛНЯТЬ**:

1 Выбираем вершину *v* из *candidates* и добавляем её в *compsub*

2 Формируем *new\_candidates* и *new\_not*, удаляя из *candidates* и *not* вершины, не СОЕДИНЕННЫЕ с *v*

3 **ЕСЛИ** *new\_candidates* и *new\_not* пусты

4 **ТО** *compsub* – клика

5 **ИНАЧЕ** рекурсивно вызываем *extend* (*new\_candidates*, *new\_not*)

6 Удаляем v из *compsub* и *candidates*, и помещаем в *not*

1. Техническое описание программного продукта.

Алгоритм реализован на языке программирования C#. Применены классы «Algorithms.cs», «EdgeGraph», «VertexGraph», «Graph», «Form1.cs», «Form1.Designer.cs»,«Programm.cs».

Были использованы:

* Объект класса «Graph» graph, содержащий в себе список вершин, представимых объектами класса «VertexGraph» и список ребер, представимых объектами класса «EdgeGraph».
* Конструкторы класса «Graph»:
* public Graph() – создает пустые список вершин и список ребер.
* public Graph(string InputText) – создает и инициализирует список вершин и список ребер.
* Методы класса «Graph»:
* public void AddVertex(VertexGraph vertex) – добавляет вершину в список вершин.
* public void AddEdge(VertexGraph from, VertexGraph to) – добавляет ребро в список ребер с весом 1.
* public void AddEdge(VertexGraph from, VertexGraph to, int weight) – добавляет ребро в список ребер с заданным весом.
* public int [,] GetMatrix() – строит матрицу смежности и возвращает её.
* public List<VertexGraph> GetVertices() – создает копию списка вершин графа и возвращает её.
* public List<EdgeGraph> GetEdges() - создает копию списка ребер графа и возвращает её.
* public List<VertexGraph> GetVertexLists(VertexGraph vertex) – возвращает список вершин смежных с вершиной vertex.
* Свойства класса «Graph»:
* public int VertexCount – возвращает кол-во вершин в списке вершин.
* public int EdgesCount – возвращает кол-во ребер в списке ребер.
* Конструктор класса «VertexGraph»:
* public VertexgGraph(int number) – задает номер вершины.
* Свойства класса «VertexGraph»:
* public int Number – возвращает или задает номер вершины.
* Методы класса «VertexGraph»:
* public override string ToString() – переводит номер вершины из типа int в тип string.
* Конструктор класса «EdgesGraph»:
* public EdgeGraph(VertexGraph from, VertexGraph to,int weight = 1) – задает ребро в виде «откуда - куда - вес ребра», (если не задан параметр weight, то по умолчанию вес ребра равен 1)
* Свойства класса «EdgesGraph»:
* public VertexGraph From { get; set; } – задает или возвращает вершину вида «откуда».
* public VertexGraph To { get; set; } – задает или возвращает вершину вида «куда».
* public int Weight { get; set; } - задает или возвращает вес ребра.
* Методы класса «EdgesGraph»:
* public override string ToString() – возвращает строку вида   
  (a; b; weight)
* Методы класса «Algorithm»:
* public void BronKerbosh(List<VertexGraph> R, List<VertexGraph> P, List<VertexGraph> X, Graph graph, List<List<VertexGraph>> MaxCliques) – возвращает список списков вершин MaxCliques – в котором находятся наибольшие клики графа. Здесь, R -множество compsub, P – множество candidates, X – множество not.

1. Инструкция по эксплуатации

Инструкция по эксплуатации такая же, как и в первой задаче.

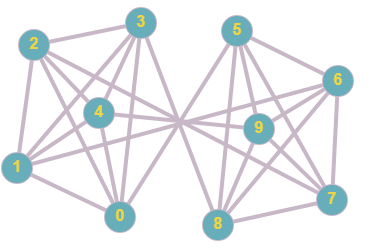
1. Набор графов для тестирования

Рисунок 24 - Первый граф для тестирования

Матрица смежности графа:

0, 1, 1, 1, 1, 1, 0, 0, 0, 0,

1, 0, 1, 1, 1, 0, 1, 0, 0, 0,

1, 1, 0, 1, 1, 0, 0, 1, 0, 0,

1, 1, 1, 0, 1, 0, 0, 0, 1, 0,

1, 1, 1, 1, 0, 0, 0, 0, 0, 1,

1, 0, 0, 0, 0, 0, 1, 1, 1, 1,

0, 1, 0, 0, 0, 1, 0, 1, 1, 1,

0, 0, 1, 0, 0, 1, 1, 0, 1, 1,

0, 0, 0, 1, 0, 1, 1, 1, 0, 1,

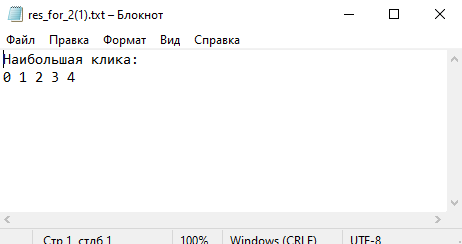
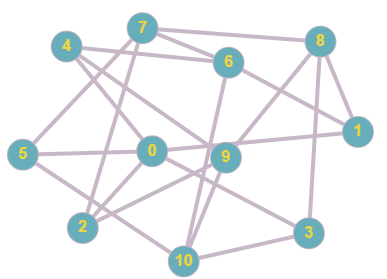
0, 0, 0, 0, 1, 1, 1, 1, 1, 0,

Рисунок 26 - Второй граф для тестирования

Рисунок 25 - Результат работы программы на первом графе

Матрица смежности графа:

0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,

1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,

1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,

1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,

1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,

1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,

0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,

0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,

0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,

0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1,

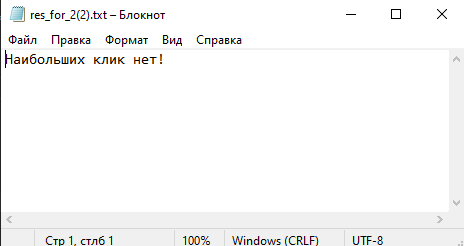
0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,

Рисунок 27 - Результат работы программы на втором графе

**Третья задача**

1. Математическая постановка задачи

Рассмотрим ориентированный связный граф, у которого:

• Существует ровно одна вершина, в которую не входит ни одна дуга (источник, s);

• Существует ровно одна вершина, из которой не выходит ни одна дуга (сток, t). Потоковой сетью называется граф указанного типа, у которого каждой дуге 𝑒 сопоставлено положительное число 𝑐(𝑒), называемое пропускной способностью дуги 𝑒.

• Множество всех дуг, входящих в вершину 𝑖 - 𝐴𝑖 + ;

• Множество всех дуг, выходящих из вершины 𝑖 - 𝐴𝑖 − .

Дано: связный неориентированный граф без петель,. Необходимо определить максимальный поток, который можно перевезти в этой сети.

1. Описание алгоритма решения

Алгоритм Эдмондса-Карпа является реализацией метода Форда-Фалкерсона, в которой в качестве дополняющего пути выбирается кратчайший по рёбрам путь в остаточной сети (длины всех рёбер равны 1).

1. Положим все потоки равными нулю. Остаточная сеть изначально совпадает с исходной сетью.
2. В остаточной сети находим кратчайший путь из источника в сток. Если такого пути нет, останавливаемся.
3. Пускаем через найденный путь (он называется увеличивающим путём или увеличивающей цепью) максимально возможный поток:
4. На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью cmin.
5. Для каждого ребра на найденном пути увеличиваем поток на cmin, а в противоположном ему — уменьшаем на cmin.
6. Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.

4. Возвращаемся на шаг 2.

3. Техническое описание программного продукта

Алгоритм реализован на языке программирования C#. Применены классы «Algorithms.cs», «EdgeGraph», «VertexGraph», «Graph», «Form1.cs», «Form1.Designer.cs»,«Programm.cs».

Были использованы:

* Объект класса «Graph» graph, содержащий в себе список вершин, представимых объектами класса «VertexGraph» и список ребер, представимых объектами класса «EdgeGraph».
* Конструкторы класса «Graph»:
* public Graph() – создает пустые список вершин и список ребер.
* public Graph(string InputText) – создает и инициализирует список вершин и список ребер.
* Методы класса «Graph»:
* public void AddVertex(VertexGraph vertex) – добавляет вершину в список вершин.
* public void AddEdge(VertexGraph from, VertexGraph to) – добавляет ребро в список ребер с весом 1.
* public void AddEdge(VertexGraph from, VertexGraph to, int weight) – добавляет ребро в список ребер с заданным весом.
* public int [,] GetMatrix() – строит матрицу смежности и возвращает её.
* public List<VertexGraph> GetVertices() – создает копию списка вершин графа и возвращает её.
* public List<EdgeGraph> GetEdges() - создает копию списка ребер графа и возвращает её.
* public List<VertexGraph> GetVertexLists(VertexGraph vertex) – возвращает список вершин смежных с вершиной vertex.
* Свойства класса «Graph»:
* public int VertexCount – возвращает кол-во вершин в списке вершин.
* public int EdgesCount – возвращает кол-во ребер в списке ребер.
* Конструктор класса «VertexGraph»:
* public VertexgGraph(int number) – задает номер вершины.
* Свойства класса «VertexGraph»:
* public int Number – возвращает или задает номер вершины.
* Методы класса «VertexGraph»:
* public override string ToString() – переводит номер вершины из типа int в тип string.
* Конструктор класса «EdgesGraph»:
* public EdgeGraph(VertexGraph from, VertexGraph to,int weight = 1) – задает ребро в виде «откуда - куда - вес ребра», (если не задан параметр weight, то по умолчанию вес ребра равен 1)
* Свойства класса «EdgesGraph»:
* public VertexGraph From { get; set; } – задает или возвращает вершину вида «откуда».
* public VertexGraph To { get; set; } – задает или возвращает вершину вида «куда».
* public int Weight { get; set; } - задает или возвращает вес ребра.
* Методы класса «EdgesGraph»:
* public override string ToString() – возвращает строку вида   
  (a; b; weight)
* Методы класса «Algorithm»:

public int EdmondsKarp(VertexGraph I, VertexGraph S, Graph graph, int[,] capacity, out int[,] legalFlows). Где I – исток, S – сток, capacity -матрица стоимостей, legalFlows – матрица кратчайших расстояний. Возвращает максимальный поток в данной сети.

int bfs (VertexGraph I,VertexGraph S, out int[] parent,Graph graph, int[,] capacity, int[,] legalFlows) – обход в ширину.

4. Инструкция по эксплуатации

Инструкция по эксплуатации такая же, как и в первой задаче.

По умолчанию исток – это первая вершина, сток – последняя.

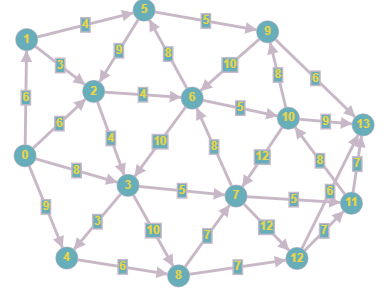
1. Набор графов для тестирования

Рисунок 28 – Первый граф для тестирования

Матрица смежности графа

0, 6, 6, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 3, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 4, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 3, 0, 0, 5, 10, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0,

0, 0, 9, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0,

0, 0, 0, 10, 0, 8, 0, 0, 0, 0, 5, 0, 0, 0,

0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 5, 12, 0,

0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 7, 0,

0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0, 6,

0, 0, 0, 0, 0, 0, 0, 12, 0, 8, 0, 0, 0, 9,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 7,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 6,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

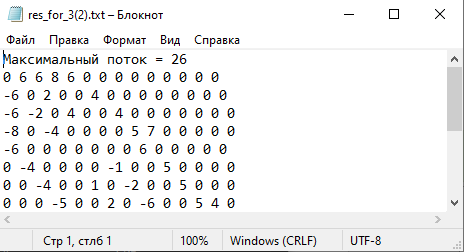


Рисунок 29 - Результат работы программы на первом графе

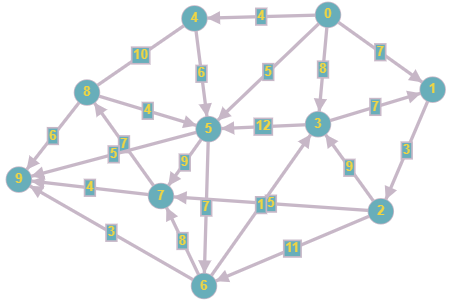


Рисунок 28 – Второй граф для тестирования

Матрица смежности графа.

0, 7, 0, 8, 4, 5, 0, 0, 0, 0,

0, 0, 3, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 9, 0, 0, 11, 5, 0, 0,

0, 7, 0, 0, 0, 12, 0, 0, 0, 0,

0, 0, 0, 0, 0, 6, 0, 0, 10, 0,

0, 0, 0, 0, 0, 0, 7, 9, 0, 5,

0, 0, 0, 1, 0, 0, 0, 8, 0, 3,

0, 0, 0, 0, 0, 0, 0, 0, 7, 4,

0, 0, 0, 0, 10, 4, 0, 0, 0, 6,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

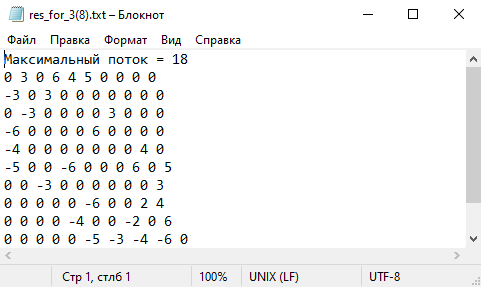


Рисунок 29 – Результат работы программы на втором графе

**Четвертая задача**

1. Математическая постановка задачи

Максимальным независимым множеством, максимальным устойчивым множеством, или максимальным стабильным множеством называется независимое множество, не являющееся подмножеством другого независимого множества. То есть это такое множество вершин S, что любое ребро графа имеет хотя бы одну конечную вершину, не принадлежащую S, и любая вершина не из S имеет хотя бы одну соседнюю в S.

Самое большое по размеру максимальное независимое множество называется наибольшим независимым множеством

Дано: связный неориентированный граф без петель,. Необходимо найти наибольшее независимое множество графа.

1. Описание алгоритма решения

Алгоритм Брона — Кербоша можно использовать для нахождения максимальных по включению независимых множеств вершин, если построить дополнение к исходному графу, либо изменив условие в основном цикле (условие остановки) и формирование новых множеств new\_candidates и new\_not:

1. Условие в основном цикле: not не должно содержать ни одной вершины, НЕ СОЕДИНЕННОЙ НИ С ОДНОЙ из вершин во множестве candidates.
2. Для формирования new\_candidates и new\_not, необходимо удалять из candidates и not вершины, СОЕДИНЕННЫЕ с выбранной вершиной.

3. Техническое описание программного продукта

Алгоритм реализован на языке программирования C#. Применены классы «Algorithms.cs», «EdgeGraph», «VertexGraph», «Graph», «Form1.cs», «Form1.Designer.cs»,«Programm.cs».

Были использованы:

* Объект класса «Graph» graph, содержащий в себе список вершин, представимых объектами класса «VertexGraph» и список ребер, представимых объектами класса «EdgeGraph».
* Конструкторы класса «Graph»:
* public Graph() – создает пустые список вершин и список ребер.
* public Graph(string InputText) – создает и инициализирует список вершин и список ребер.
* Методы класса «Graph»:
* public void AddVertex(VertexGraph vertex) – добавляет вершину в список вершин.
* public void AddEdge(VertexGraph from, VertexGraph to) – добавляет ребро в список ребер с весом 1.
* public void AddEdge(VertexGraph from, VertexGraph to, int weight) – добавляет ребро в список ребер с заданным весом.
* public int [,] GetMatrix() – строит матрицу смежности и возвращает её.
* public List<VertexGraph> GetVertices() – создает копию списка вершин графа и возвращает её.
* public List<EdgeGraph> GetEdges() - создает копию списка ребер графа и возвращает её.
* public List<VertexGraph> GetVertexLists(VertexGraph vertex) – возвращает список вершин смежных с вершиной vertex.
* Свойства класса «Graph»:
* public int VertexCount – возвращает кол-во вершин в списке вершин.
* public int EdgesCount – возвращает кол-во ребер в списке ребер.
* Конструктор класса «VertexGraph»:
* public VertexgGraph(int number) – задает номер вершины.
* Свойства класса «VertexGraph»:
* public int Number – возвращает или задает номер вершины.
* Методы класса «VertexGraph»:
* public override string ToString() – переводит номер вершины из типа int в тип string.
* Конструктор класса «EdgesGraph»:
* public EdgeGraph(VertexGraph from, VertexGraph to,int weight = 1) – задает ребро в виде «откуда - куда - вес ребра», (если не задан параметр weight, то по умолчанию вес ребра равен 1)
* Свойства класса «EdgesGraph»:
* public VertexGraph From { get; set; } – задает или возвращает вершину вида «откуда».
* public VertexGraph To { get; set; } – задает или возвращает вершину вида «куда».
* public int Weight { get; set; } - задает или возвращает вес ребра.
* Методы класса «EdgesGraph»:
* public override string ToString() – возвращает строку вида   
  (a; b; weight)
* Методы класса «Algorithm»:

public void BronKerbosh\_MIS(List<VertexGraph> R, List<VertexGraph> P, List<VertexGraph> X,Graph graph,List<List<VertexGraph>> MaxIS) - возвращает список списков вершин MaxIs – в котором находятся независимые множества графа. Здесь, R -множество compsub, P – множество candidates, X – множество not.

4. Инструкция по эксплуатации

Инструкция по эксплуатации такая же, как и в первой задаче.

1. Набор графов для тестирования

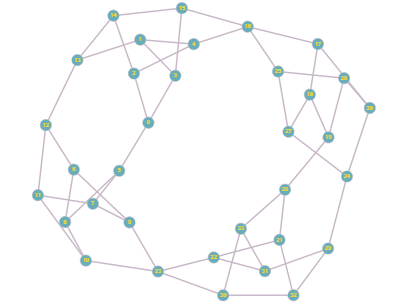


Рисунок 30 – Первый граф для тестирования

Матрица смежности графа.

0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,

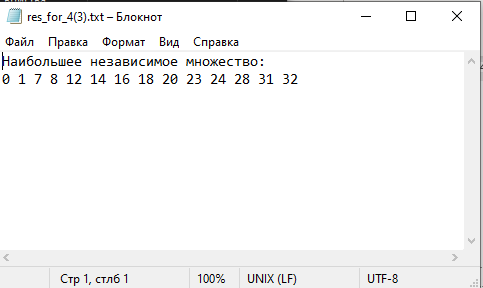


Рисунок 31 – Результат работы программы на втором графе

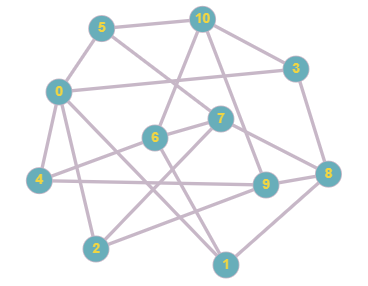
****

Рисунок 32 – Второй граф для тестирования

Матрица смежности графа.

0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,

1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,

1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,

1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,

1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,

1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,

0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,

0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,

0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,

0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1,

0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0,

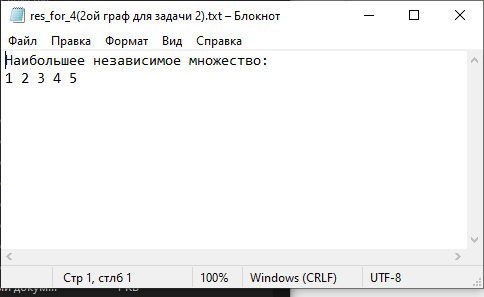


Рисунок 33 – Результат работы программы на втором графе

**Пятая задача**

1. Математическая постановка задачи.

Дан ориентированный или неориентированный взвешенный граф  с  вершинами. Требуется найти значения всех величин d_{ij} — длины кратчайшего пути из вершины i в вершину j.

Предполагается, что граф не содержит циклов отрицательного веса (тогда ответа между некоторыми парами вершин может просто не существовать — он будет бесконечно маленьким).

1. Описание алгоритма решения

Ключевая идея алгоритма — разбиение процесса поиска кратчайших путей на **фазы**.

Перед k-ой фазой (k = 1 \ldots n) считается, что в матрице расстояний d[][] сохранены длины таких кратчайших путей, которые содержат в качестве внутренних вершин только вершины из множества \{ 1, 2, \ldots, k-1 \} (вершины графа мы нумеруем, начиная с единицы).

Иными словами, перед k-ой фазой величина d[i][j] равна длине кратчайшего пути из вершины i в вершину j, если этому пути разрешается заходить только в вершины с номерами, меньшими k (начало и конец пути не считаются).

Легко убедиться, что чтобы это свойство выполнилось для первой фазы, достаточно в матрицу расстояний d[][] записать матрицу смежности графа: d[i][j] = g[i][j] — стоимости ребра из вершины i в вершину j. При этом, если между какими-то вершинами ребра нет, то записать следует величину "бесконечность" \infty. Из вершины в саму себя всегда следует записывать величину 0, это критично для алгоритма.

Пусть теперь мы находимся на k-ой фазе, и хотим **пересчитать** матрицу d[][] таким образом, чтобы она соответствовала требованиям уже для k+1-ой фазы. Зафиксируем какие-то вершины i и j. У нас возникает два принципиально разных случая:

* Кратчайший путь из вершины i в вершину j, которому разрешено дополнительно проходить через вершины \{ 1, 2, \ldots, k \}, **совпадает** с кратчайшим путём, которому разрешено проходить через вершины множества \{ 1, 2, \ldots, k-1 \}.

В этом случае величина d[i][j] не изменится при переходе с k-ой на k+1-ую фазу.

* "Новый" кратчайший путь стал **лучше** "старого" пути.

Это означает, что "новый" кратчайший путь проходит через вершину k. Сразу отметим, что мы не потеряем общности, рассматривая далее только простые пути (т.е. пути, не проходящие по какой-то вершине дважды).

Тогда заметим, что если мы разобьём этот "новый" путь вершиной k на две половинки (одна идущая i \Rightarrow k, а другая — k \Rightarrow j), то каждая из этих половинок уже не заходит в вершину k. Но тогда получается, что длина каждой из этих половинок была посчитана ещё на k-1-ой фазе или ещё раньше, и нам достаточно взять просто сумму d[i][k] + d[k][j], она и даст длину "нового" кратчайшего пути.

**Объединяя** эти два случая, получаем, что на k-ой фазе требуется пересчитать длины кратчайших путей между всеми парами вершин i и j следующим образом:

new\_d[i][j] = min (d[i][j], d[i][k] + d[k][j]);

Таким образом, вся работа, которую требуется произвести на k-ой фазе — это перебрать все пары вершин и пересчитать длину кратчайшего пути между ними. В результате после выполнения n-ой фазы в матрице расстояний d[i][j] будет записана длина кратчайшего пути между i и j, либо \infty, если пути между этими вершинами не существует.

3. Техническое описание программного продукта

Алгоритм реализован на языке программирования C#. Применены классы «Algorithms.cs», «EdgeGraph», «VertexGraph», «Graph», «Form1.cs», «Form1.Designer.cs»,«Programm.cs».

Были использованы:

* Объект класса «Graph» graph, содержащий в себе список вершин, представимых объектами класса «VertexGraph» и список ребер, представимых объектами класса «EdgeGraph».
* Конструкторы класса «Graph»:
* public Graph() – создает пустые список вершин и список ребер.
* public Graph(string InputText) – создает и инициализирует список вершин и список ребер.
* Методы класса «Graph»:
* public void AddVertex(VertexGraph vertex) – добавляет вершину в список вершин.
* public void AddEdge(VertexGraph from, VertexGraph to) – добавляет ребро в список ребер с весом 1.
* public void AddEdge(VertexGraph from, VertexGraph to, int weight) – добавляет ребро в список ребер с заданным весом.
* public int [,] GetMatrix() – строит матрицу смежности и возвращает её.
* public List<VertexGraph> GetVertices() – создает копию списка вершин графа и возвращает её.
* public List<EdgeGraph> GetEdges() - создает копию списка ребер графа и возвращает её.
* public List<VertexGraph> GetVertexLists(VertexGraph vertex) – возвращает список вершин смежных с вершиной vertex.
* Свойства класса «Graph»:
* public int VertexCount – возвращает кол-во вершин в списке вершин.
* public int EdgesCount – возвращает кол-во ребер в списке ребер.
* Конструктор класса «VertexGraph»:
* public VertexgGraph(int number) – задает номер вершины.
* Свойства класса «VertexGraph»:
* public int Number – возвращает или задает номер вершины.
* Методы класса «VertexGraph»:
* public override string ToString() – переводит номер вершины из типа int в тип string.
* Конструктор класса «EdgesGraph»:
* public EdgeGraph(VertexGraph from, VertexGraph to,int weight = 1) – задает ребро в виде «откуда - куда - вес ребра», (если не задан параметр weight, то по умолчанию вес ребра равен 1)
* Свойства класса «EdgesGraph»:
* public VertexGraph From { get; set; } – задает или возвращает вершину вида «откуда».
* public VertexGraph To { get; set; } – задает или возвращает вершину вида «куда».
* public int Weight { get; set; } - задает или возвращает вес ребра.
* Методы класса «EdgesGraph»:
* public override string ToString() – возвращает строку вида   
  (a; b; weight)
* Методы класса «Algorithm»:

public int[,] Floyd\_Warshall(Graph graph, int[,] next) – функция, которая возвращает матрицу кратчайших путей и задает матрицу next – для восстановления кратчайшего пути от одной вершины к другой.

public void Path(VertexGraph u, VertexGraph v,int[,] next,List<VertexGraph> path) – функция, которая возвращает кратчайший путь от вершины u до вершины v. Сам путь хранится в списке вершин path.

1. Инструкция по эксплуатации

Инструкция по эксплуатации такая же, как и в первой задаче.

По умолчанию находится кратчайший путь от первой вершины до последней.

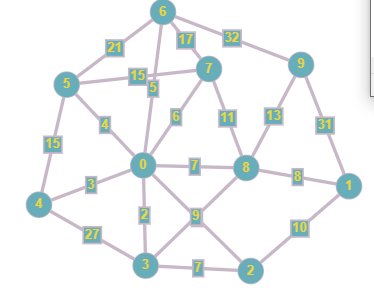
1. Набор графов для тестирования

Рисунок 34 – Первый граф для тестирования

Матрица смежности графа.

0, 0, 1, 2, 3, 4, 5, 6, 7, 0,

0, 0, 10, 0, 0, 0, 0, 0, 8, 31,

1, 10, 0, 7, 0, 0, 0, 0, 0, 0,

2, 0, 7, 0, 27, 0, 0, 0, 9, 0,

3, 0, 0, 27, 0, 15, 0, 0, 0, 0,

4, 0, 0, 0, 15, 0, 21, 15, 0, 0,

5, 0, 0, 0, 0, 21, 0, 17, 0, 32,

6, 0, 0, 0, 0, 15, 17, 0, 11, 0,

7, 8, 0, 9, 0, 0, 0, 11, 0, 13,

0, 31, 0, 0, 0, 0, 32, 0, 13, 0,

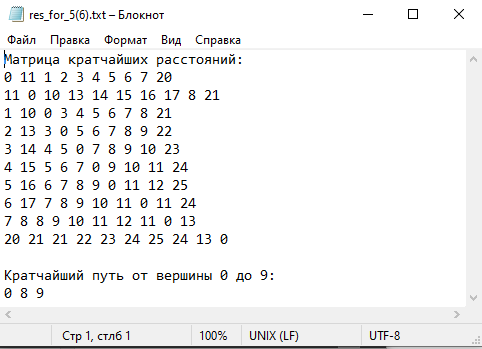
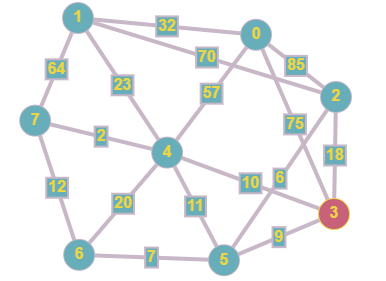
****

Рисунок 36 – Второй граф для тестирования

Рисунок 35 – Результат работы программы на первом графе

Матрица смежности графа

0, 32, 85, 75, 57, 0, 0, 0,

32, 0, 70, 0, 23, 0, 0, 64,

85, 70, 0, 18, 0, 6, 0, 0,

75, 0, 18, 0, 10, 9, 0, 0,

57, 23, 0, 10, 0, 11, 20, 2,

0, 0, 6, 9, 11, 0, 7, 0,

0, 0, 0, 0, 20, 7, 0, 12,

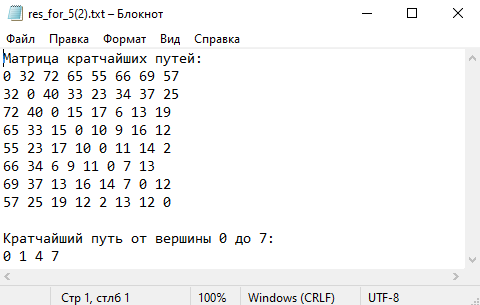
****0, 64, 0, 0, 2, 0, 12, 0,

Рисунок 37 – Результат работы программы на втором графе

**Cписок использованной литературы**

1. Система вопросов и ответов о программировании [Электронный ресурс]. – Режим доступа: https://ru.stackoverflow.com. – (Дата обращения: 20.05.2020).
2. Система вопросов и ответов о программировании [Электронный ресурс]. – Режим доступа: http://www.e-maxx-ru.1gb.ru – (Дата обращения: 20.05.2020).