Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра АОИ

**Алгоритмы на графах**

**ОТЧЕТ**

ПО РЕЗУЛЬТАТАМ

Учебной практики: по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

|  |  |
| --- | --- |
|  | Обучающийся гр. з-427П5-4 .  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лобин М.А. .  (подпись) (И.О. Фамилия)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата) |
|  |  |
| **\_\_\_\_\_\_\_\_\_**  (оценка) | Руководитель практики от Университета:  ст. преподаватель каф. АОИ .  (должность, ученая степень, звание)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пермякова Н.В. . (подпись) (И.О. Фамилия)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата) |

Томск 2020

# Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра АОИ

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Зав. кафедрой АОИ  Сидоров А.А.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на учебную практику: по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

студенту гр. з-427П5-4 факультета дистанционного обучения

Лобину Максиму Александровичу .

(Ф.И.О студента)

1. Тема практики: Алгоритмы на графах
2. Цель практики: формирование первичных умений и навыков программирования
3. Задачи практики: Реализация алгоритма Форда – Беллмана. Изучить алгоритм для поиска кратчайшего пути, разработать приложение на языке C#.Отладить и протестировать работу приложения.
4. Вариант 15. Реализация алгоритма Форда.
5. Сроки прохождения практики: 29.06.2020 - 25.07.2020.

**Совместный рабочий график (план) проведения практики**

|  |  |  |
| --- | --- | --- |
| № п/п | Перечень заданий | Сроки выполнения |
| 1 | Изучение алгоритма Форда – Беллмана | 29.06 – 01.07.2020 |
| 2 | Разработка интерфейса | 02.07 – 03.07.2020 |
| 3 | Реализация алгоритма (наброски в конс. приложении) | 06.07 – 10.07.2020 |
| 4 | Создание WinForm приложения | 13.07 – 17.07.2020 |
| 5 | Заполнение отчета и дневника о практике | 20.07 – 24.07.2020 |

Дата выдачи: «29» июня 2020 г.

Руководитель практики от университета

|  |  |  |
| --- | --- | --- |
| ст. преподаватель каф. АОИ .  (должность) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Подпись) | Пермякова Н.В. .  (Ф.И.О.) |

Задание принял к исполнению «29» июня 2020 г.

|  |  |  |
| --- | --- | --- |
| Студент гр. . з-427П5-4 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Подпись) | Лобин М.А. .  (Ф.И.О.) |

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc51883922)

[1 ОПИСАНИЕ АЛГОРИТМА 5](#_Toc51883923)

[2 РЕАЛИЗАЦИЯ АЛГОРИТМА 9](#_Toc51883924)

[2.1 Библиотека классов «clsГраф» 9](#_Toc51883925)

[2.2 Библиотека классов «ctrlУправлениеГрафами» 11](#_Toc51883926)

[2.3 Проект Windows Form «FordsAlgorithm» 13](#_Toc51883927)

[3 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ 15](#_Toc51883928)

[ЗАКЛЮЧЕНИЕ 21](#_Toc51883929)

[СПИСОК ЛИТЕРАТЕРЫ 22](#_Toc51883930)

[ПРИЛОЖЕНИЕ А 23](#_Toc51883931)

[ПРИЛОЖЕНИЕ Б 28](#_Toc51883932)

**ВВЕДЕНИЕ**

Графы представляют собой наиболее абстрактную структуру, с которой приходится сталкиваться в теории ЭВМ. Графы используются для описания алгоритмов автоматического проектирования, в диаграммах машины конечных состояний, при решении задач маршрутизации потоков и т.д. Любая система, предполагающая наличие дискретных состояний или наличие узлов и переходов между ними может быть описана графом.

Граф представляет собой непустое множество точек и множество отрезков, оба конца которых принадлежат заданному множеству точек. При изображении графов на рисунках или схемах отрезки могут быть прямолинейными или криволинейными; длины отрезков и расположение точек произвольны.

Примерами графов могут служить схема метрополитена, схемы железных или шоссейных дорог, структурные формулы молекул, планы выставок и т. д., словом, схемы и планы (или карты) без указания масштабов, показывающие лишь связи между принадлежащими им объектами.

1. **ОПИСАНИЕ АЛГОРИТМА**

Алгоритм Беллмана-Форда позволяет решить задачу о кратчайшем пути из одной вершины в общем случае, когда вес каждого из ребер может быть отрицательным. Для заданного взвешенного ориентированного графа G = (V, Е) с истоком s и весовой функцией w : Е —» R алгоритм Беллмана-Форда возвращает логическое значение, указывающее на то, содержится ли в графе цикл с отрицательным весом, достижимый из истока. Если такой цикл существует, в алгоритме указывается, что решения не существует. Если же таких циклов нет, алгоритм выдает кратчайшие пути и их вес.

Предположим, что вершина 1 является вершиной-входом, и требуется найти длины кратчайших путей от вершины 1 до каждой другой вершины графа. Для этого алгоритма дуговые расстояния могут быть как положительными, так и отрицательными, но не должно быть циклов отрицательной длины. Предположим, что если в графе отсутствует та или иная дуга, то её вес равен бесконечно большому числу. Основная идея алгоритма Беллмана – Форда состоит в том, чтобы сначала найти длины кратчайших путей, при условии, что пути содержат не более двух дуг, не более трех дуг и т.д. Кратчайший путь, при условии, что он содержит не более h дуг, будем называть кратчайшим путем в графе. Согласно идее Беллмана -39 Форда, Р. Прим предложил алгоритм нахождения длин кратчайших путей, ведущих из произвольной вершины графа во все остальные его вершины (в которые есть пути из вершины-входа), состоящий в присвоении вершинам некоторых потенциалов. Признаком окончания алгоритма является остановка процесса изменения потенциалов.

Алгоритм, предложенный Р. Примом, состоит в выполнении следующих операций:

1. Вершине – входу присваивается потенциал ; остальным вершинам – потенциал ∞, i = 1 … n, где n – число вершин в рассматриваемом графе.
2. Для каждой вершины i, смежной с вершиной j из массива вершин L, находим

и проверяем неравенство

|  |  |
| --- | --- |
|  | (1.1) |

Если неравенство выполняется, то

|  |  |
| --- | --- |
|  | (1.2) |

Вершину i заносим в массив

1. Проверяем массив L1 ≠ Ш. Если неравенство выполняется L := L1,

осуществляем переход к п. 2.

1. Конец алгоритма нахождения кратчайших путей.

Рассмотрим работу алгоритма на примере.

**Пример**

Допустим, имеется граф (Рисунок 1.1) и представленный список дуг (Таблица 1.1) ). [1. Стр. 39]

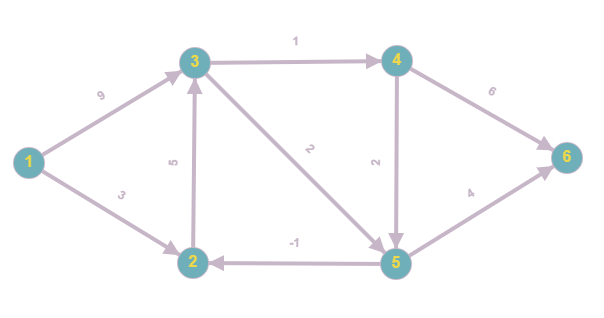


Рисунок 1.1 – Пример графа

Таблица 1.1 Список дуг

|  |  |
| --- | --- |
| (1,2) – вес 3 | (4,6) – вес 6 |
| (1,3) – вес 9 | (3,5) – вес 2 |
| (2,3) – вес 5 | (5,2) – вес -1 |
| (3,4) – вес 1 | (5,6) – вес 4 |
| (4,5) – вес 2 | (6,0) – вес 0 |

Результат работы алгоритма сведен в таблицу 1.2.

Таблица 1.2 Результат работы алгоритма

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Шаг вычислений** | **Массив** | **Потенциалы вершин** | | | | | | **Массив** |
| **1** | **2** | **3** | **4** | **5** | **6** |
| 0 | Ш | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | 1 |
| 1 | 1 | 0 | 3 | 9 | ∞ | ∞ | ∞ | 2, 3 |
| 2 | 2, 3 | 0 | 3 | 8 | 10 | 11 | ∞ | 3, 4, 5 |
| 3 | 3, 4, 5 | 0 | 3 | 8 | 9 | 10 | 15 | 4, 5, 6 |
| 4 | 4, 5, 6 | 0 | 3 | 8 | 9 | 10 | 14 | 6 |
| 5 | 6 | 0 |  | 8 | 9 | 10 | 14 | Ш |

На нулевом шаге производится инициализация согласно п.1 алгоритма. Далее на первом шаге вычислений рассматриваются вершины, смежные с вершиной-входом, потенциалы рассматриваемых вершин становятся равными длине, соединяющей их дуги, если таковое существует, в противном случае – потенциал вершины остается равным ∞.

На втором шаге рассматриваем вершины, достижимые из вершины входа при помощи пути, состоящего из двух дуг. Таковыми являются:

1. 1 – 2 – 3: потенциал вершины 3 изменяется ;
2. 1 – 3 – 4: потенциал вершины 4 изменяется ;
3. 1 – 3 – 5: потенциал вершины 5 изменяется .

Третий шаг:

1. 1 – 3 – 4 – 6: потенциал вершины 6 изменяется ;
2. 1 – 2 – 3 – 5: потенциал вершины 5 изменяется ;
3. 1 – 3 – 5 – 2: потенциал вершины 2 не изменится ();
4. 1 – 3 – 4 – 5: потенциал вершины 5 не изменится ();
5. 1 – 3 – 5 – 6: потенциал вершины 6 изменится .

Четвертый шаг:

1. 1 – 2 – 3 – 5 – 6: потенциал вершины 6 изменится ;
2. 1 – 2 – 3 – 4 – 5: потенциал вершины 5 не изменится ();
3. 1 – 2 – 3 – 5 – 2: потенциал вершины 2 не изменится ();
4. 1 – 3 – 5 – 2 – 3: потенциал вершины 3 не изменится ();
5. 1 – 3 – 4 – 5 – 2: потенциал вершины 2 не изменится ().

Пятый шаг:

1. 1 – 3 – 5 – 2 – 3 – 5: потенциал вершины 5 не изменится ();
2. 1 – 3 – 5 – 2 – 3 – 4: потенциал вершины 4 не изменится ();
3. 1 – 3 – 4 – 5 – 2 – 3: потенциал вершины 3 не изменится ();
4. 1 – 2 – 3 – 4 – 5 – 2: потенциал вершины 2 не изменится ();
5. 1 – 2 – 3 – 4 – 5 – 6: потенциал вершины 6 не изменится ();
6. 1 – 2 – 3 – 5 – 2 – 3: потенциал вершины 3 не изменится ().

На пятом шаге вычислений не произошло изменений ни в одном из

потенциалов вершин графа, делаем вывод об окончании работы алгоритма.

Восстановим кратчайший путь из вершины 6 в вершину 1. Для этого

для каждой вершины j определим такую вершину i, что будет выполняться

правило

|  |  |
| --- | --- |
|  | (1.3) |

Следуя правилу, последовательно находим кратчайший путь:

вершина 6: i = 5;

вершина 5: i = 3;

вершина 3: i = 2;

вершина 2: i = 1.

Кратчайшим путем (рисунок 1.2) является путь 1 – 2 – 3 – 5 – 6.

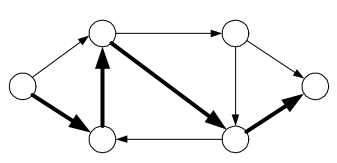


Рисунок 1.2 – Кратчайший путь.

1. **РЕАЛИЗАЦИЯ АЛГОРИТМА**

Алгоритм реализован по средствам Windows Form приложения написанного на языке C#. В качестве IDE использована Visual Studio 2017.

Решение разделено на три проекта:

* Библиотека классов «clsГраф», представляющая объектную модель графа, как объекта;
* Библиотека классов «ctrlУправлениеГрафами», представляющая пользовательский элемент управления, для обработки массива графов (добавление, удаление, установка параметров);
* Проект Windows Form «FordsAlgorithm», представляет главную экранную форму.
  1. Библиотека классов «clsГраф»

Полный листинг библиотеки представлен в приложении А.

Класс «Граф»

* Конструкторы

|  |  |
| --- | --- |
| Граф(string, int = 0) | Инициализирует новый экземпляр класс Граф.  Параметры:  *ИмяГрафа* string– наименование графа;  *КоличествоВершин* int – указанное количество вершин, будет создано при инициализации нового экземпляра графа |
| Граф(Guid, string, int = 0) | Инициализирует новый экземпляр класс Граф.  Параметры:  *UIDГрафа* Guid – идентификатор экземпляра графа  *ИмяГрафа* string– наименование графа;  *КоличествоВершин* int – указанное количество вершин, будет создано при инициализации нового экземпляра графа |

* Поля класса

|  |  |
| --- | --- |
| ИмяГрафа string | Поле хранит имя текущего экземпляра графа. |
| UIDГрафа guid | Поле хранит идентификатор текущего экземпляра графа. |
| Вершины List<[Вершина](#КлассВершина)> | Поле хранит коллекцию Вершин текущего экземпляра графа. |
| Ребра List<[Ребро](#КлассРебро)> | Поле хранит коллекцию Ребер текущего экземпляра графа. |

* Свойства

|  |  |
| --- | --- |
| int КоличествоВершин | Получает значение, определяющее количество вершин текущего экземпляра графа. |
| int КоличествоРебер | Получает значение, определяющее количество ребер текущего экземпляра графа. |

* Методы (открытые)

|  |  |
| --- | --- |
| void ДобавитьВершину([Вершина](#КлассВершина)) | Добавляет заданную вершину в текущий экземпляр класса |
| void ДобавитьРебро([Ребро](#КлассРебро)) | Добавляет заданное ребро в текущий экземпляр класса |
| void УдалитьРебро([Ребро](#КлассРебро)) | Удаляет указанное ребро в текущем экземпляре класса |
| Вершина ВершинаПоНомеру(int) | Возвращает вершину из текущего графа по ее номеру |
| void ОчиститьГраф() | Удаляет в текущем экземпляре графа все вершины и все ребра |
| string ОпределитьКороткийПуть(int, int) | По начальной и конечной вершине, определяет кратчайший путь между ними |
| DataTable ПостороитьМатрицуСмежности() | Возвращает матрицу смежности в виде DataTable |

* Методы (закрытые)

|  |  |
| --- | --- |
| List<[Вершина](#КлассВершина)>РасчетПотенциалаВершин([Вершина](#КлассВершина), int) | Возвращает массив вершин, являющихся смежными для указанной вершины с заданной глубиной |
| void КратчайшийПуть ([Вершина](#КлассВершина), List<string>) | В указанный массив строк, записывается кратчайший путь |
| int? ОпределитьБесконечность () | Возвращает сумму весов всех ребер графа. |

Класс «Вершина»

* Конструкторы

|  |  |
| --- | --- |
| Вершина(int) | Инициализирует новый экземпляр класс Вершина.  Параметры:  *НомерВершины* int– номер создаваемой вершины |

* Поля класса

|  |  |
| --- | --- |
| int НомерВершины | Хранит номер вершины или позицию в текущем графе |
| int? Потенциал | Хранит значение ослабления при расчете кратчайшего пути |
| [Вершина](#КлассВершина) Источник | Хранит вершину, которая является наилучшим источником при прохождении кротчайшего пути |

Класс «Ребро»

* Конструкторы

|  |  |
| --- | --- |
| Ребро(int, [Вершина](#КлассВершина), [Вершина](#КлассВершина)) | Инициализирует новый экземпляр класс Ребро.  Параметры:  *Вес* int– вес ребра;  *ВекторНачало* [Вершина](#КлассВершина) – Начальная вершина ребра  *ВекторКонец* [Вершина](#КлассВершина) – Конечная вершина ребра |

* Поля класса

|  |  |
| --- | --- |
| int Вес | Хранит вес ребра |
| [Вершина](#КлассВершина) ВекторНачало | Хранит вершину начального вектора ребра |
| [Вершина](#КлассВершина) ВекторКонец | Хранит вершину конечного вектора ребра |

* 1. Библиотека классов «ctrlУправлениеГрафами»

Библиотека содержит только пользовательский элемент управления для управления массивом графов. Интерфейс элемента представлен на рисунке 2.1.

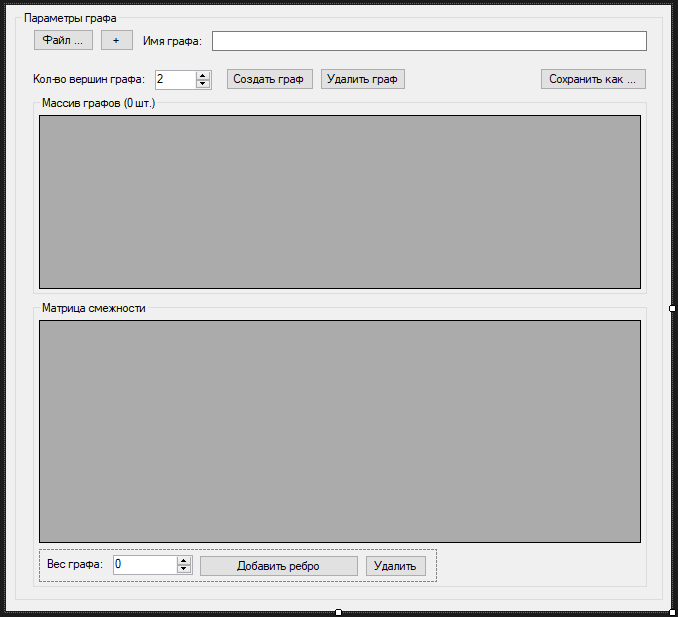


Рисунок 2.1 – Интерфейс контрола UserControlНовыйГраф

Полный листинг библиотеки представлен в приложении В.

Класс «UserControlНовыйГраф»

* Конструкторы

|  |  |
| --- | --- |
| UserControlНовыйГраф() | Инициализирует новый экземпляр класс UserControlНовыйГраф. |

* Поля (закрытые)

|  |  |
| --- | --- |
| МассивГрафовClass МассивГрафовDT | Хранит таблицу с перечнем графов. |
| List<Граф> | Хранит коллекцию графов |
| int МатрицаТекущийСтолбец | Хранит текущую позицию курсора относительно столбцов в таблице матрицы смежности |
| int МатрицаТекущаяСтрока | Хранит текущую позицию курсора относительно строк в таблице матрицы смежности |

* Свойства

|  |  |
| --- | --- |
| Guid? UIDТекущегоГрафа | Получает идентификатор текущего графа, выделенного в текущий момент в таблице графов |
| Граф ТекущийГраф | Получает объект Граф, выделенный в текущий момент в таблице графов |

* События

|  |  |
| --- | --- |
| EventHandler КоличествоРеберChanged | Инициализируется событие, при изменении количества ребер в графе |
| EventHandler МассивГрафовSelectionChanged | Инициализируется при изменении фокуса в таблице с массивом графов |
| EventHandler МассивГрафовДобавление | Инициализируется при добавлении нового графа в массив графов |
| EventHandler МассивГрафовУдаление | Инициализируется при удалении графа из массива графов |

* 1. Проект Windows Form «FordsAlgorithm»

Проект представляет собой экранную форму, содержащую элемент управления «UserControlНовыйГраф», консоль выполнения для отображения логов, кнопку запускающую поиск кратчайшего пути и выпадающие списки для выбора начальной конечной вершины. Интерфейс формы представлен на Рисунке 2.2.

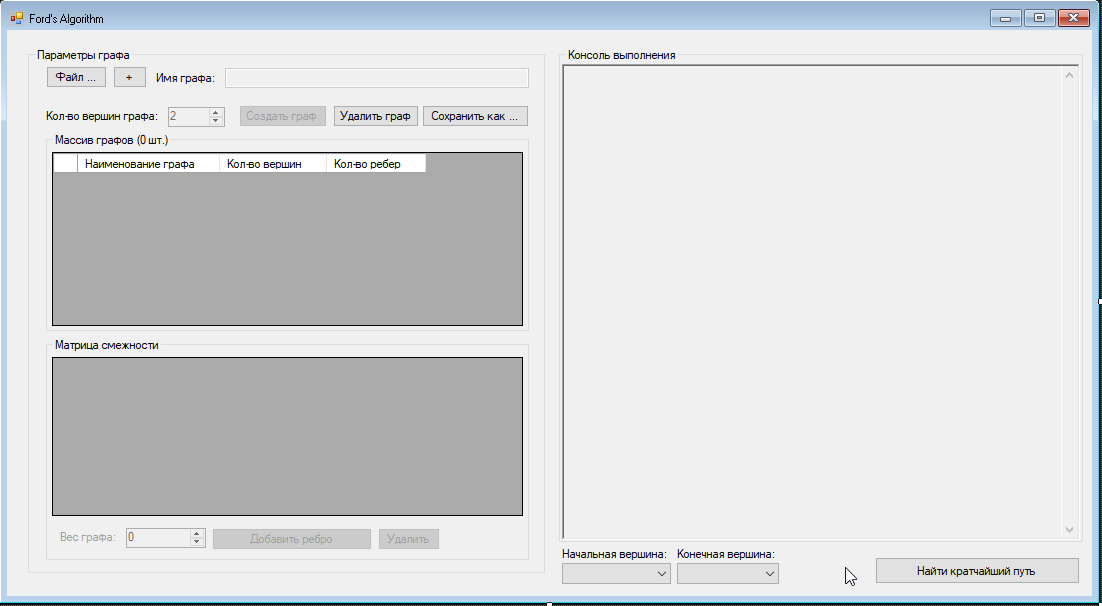


Рисунок 2.2 – Интерфейс формы

Интерфейс формы позволяет проделать следующие манипуляции:

* Создание, удаление графа;
* Настройка графа по средствам таблицы матрицы смежности;
* Сохранение графа в файл с расширением \*.graf и последующее его использование;
* Поиск кратчайшего пути в выбранном графе.

**Описание функционала**

На рисунке 2.3 представлена схема функциональности

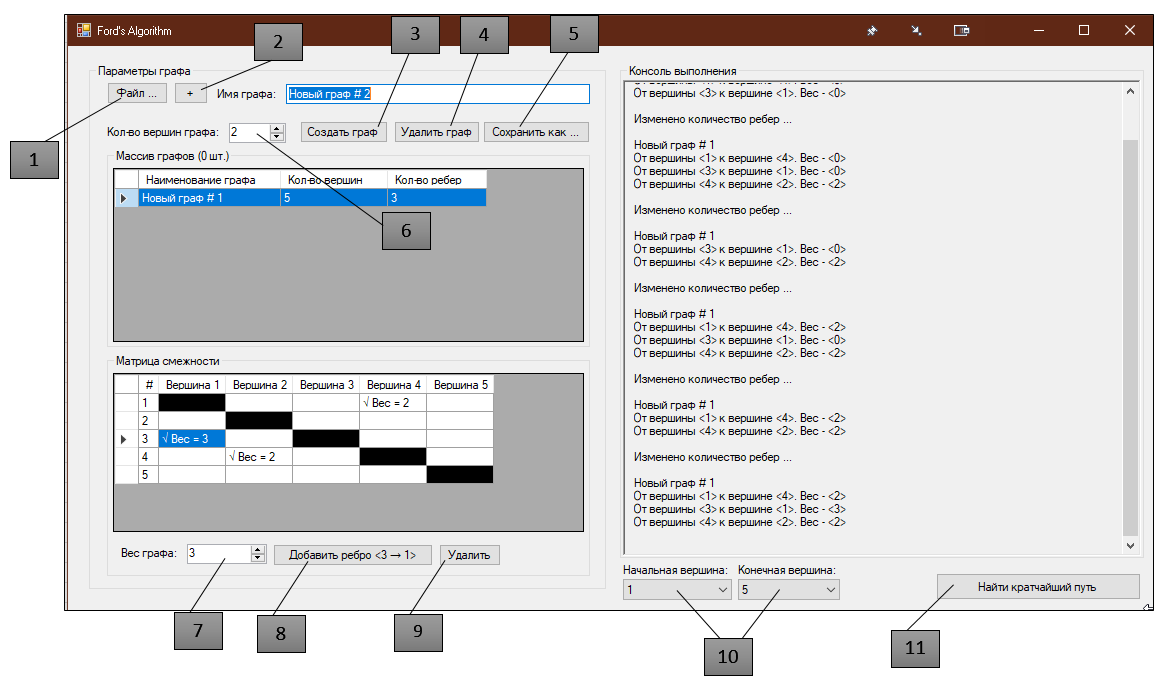


Рисунок 2.3 – Описание функционала

1. Открытие ранее сохраненного графа.
2. Добавление нового виртуального графа.
3. Создание из виртуального графа, физического, с добавлением его в массив.
4. Удаление графа из массива графов.
5. Сохранение графа в файл
6. Количество вершин в графе. Задается при создании, в последующем изменить не возможно.
7. Вес ребра при его создании
8. Создание нового ребра на пересечении выбранной ячейки в матрице смежности.
9. Удаление ребра из графа.
10. Начальная и конечная вершины для поиска кратчайшего пути.
11. Поиск кратчайшего пути.
12. **ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ**

Пример №1

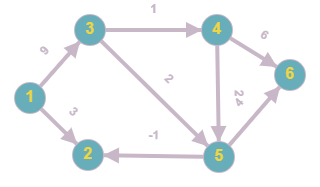


Рисунок 3.1 - Эскиз графа – Пример №1

С помощью таблицы матрицы смежности составим граф (См. рисунок 3.2).

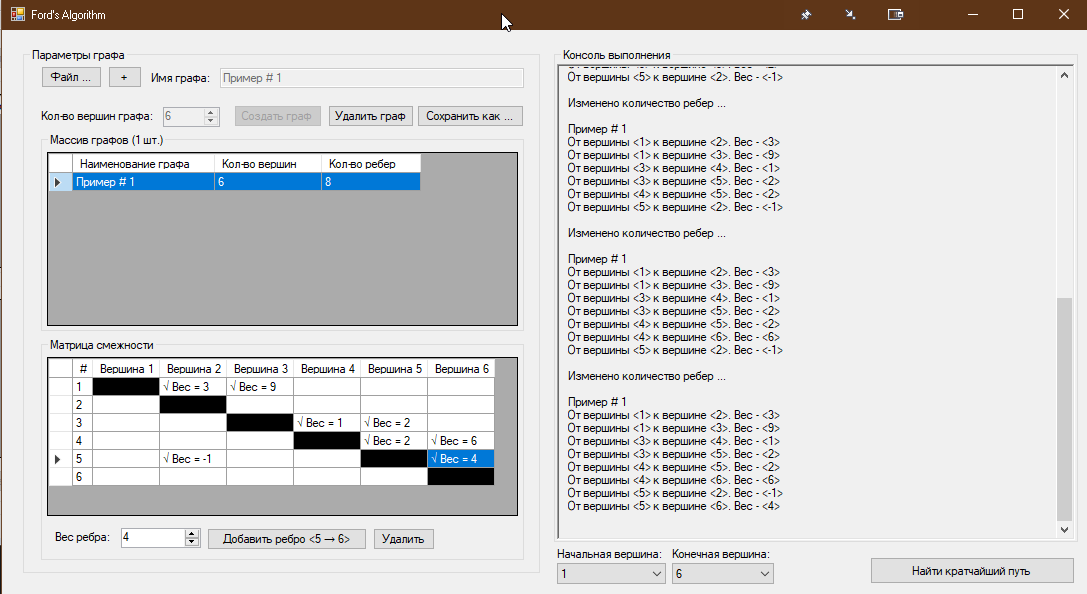


Рисунок 3.2 – Граф – Пример №1

Определим кратчайший путь из вершины 1 к вершине 6

Определим кратчайший путь из вершины 2 к вершине 5

Определим кратчайший путь из вершины 3 к вершине 6

Определим кратчайший путь из вершины 1 к вершине 4

Результат расчетов представлен на Рисунке 3.3.

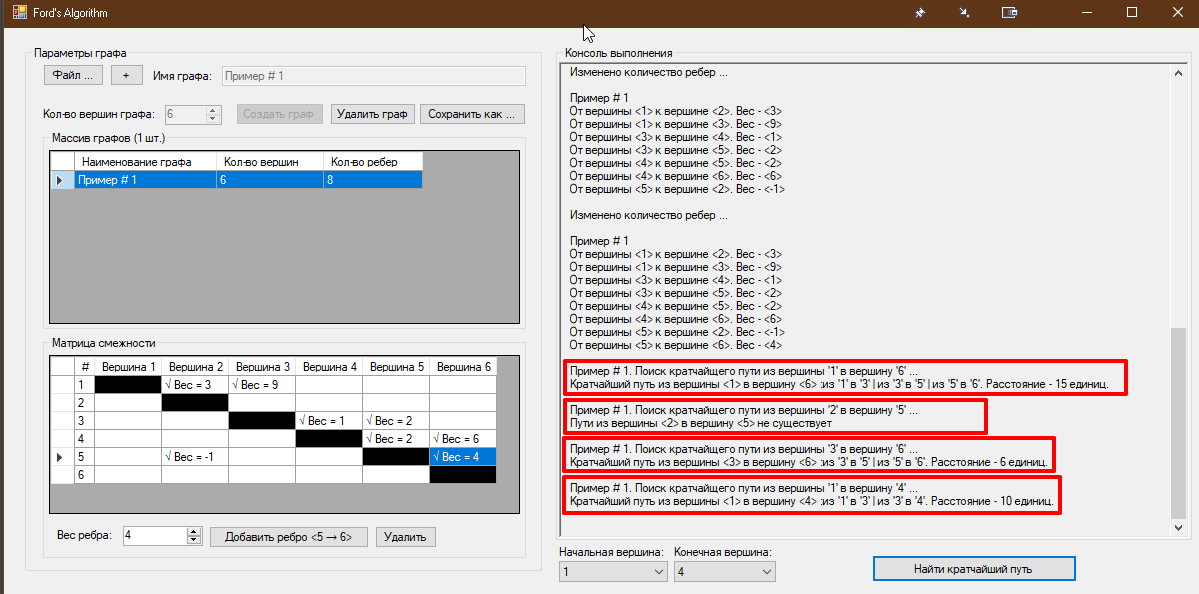


Рисунок 3.3 – Результат расчетов на графе Пример №1

Пример №2

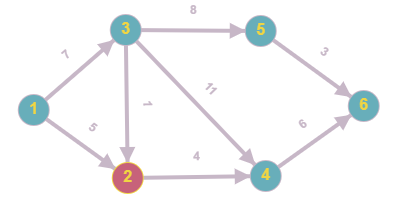


Рисунок 3.4 – Эскиз графа - Пример №2

С помощью таблицы матрицы смежности составим граф (См. рисунок 3.5).

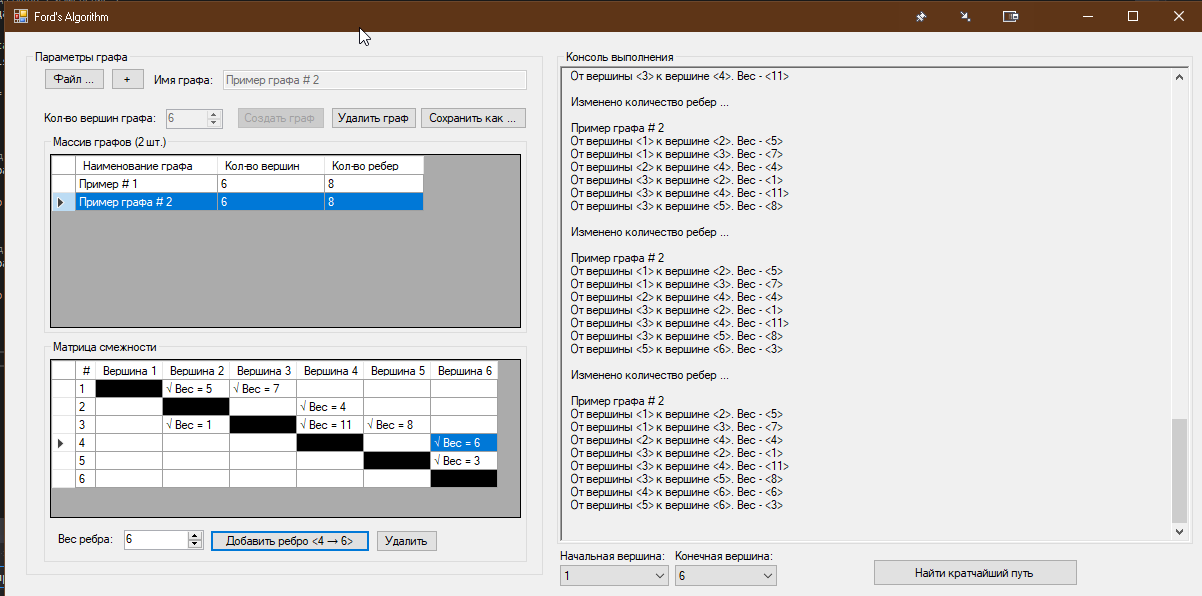


Рисунок 3.5 – Граф – Пример №2

Определим кратчайший путь из вершины 1 к вершине 6

Определим кратчайший путь из вершины 1 к вершине 4

Определим кратчайший путь из вершины 2 к вершине 5

Определим кратчайший путь из вершины 3 к вершине 6

Результат расчетов представлен на Рисунке 3.6.

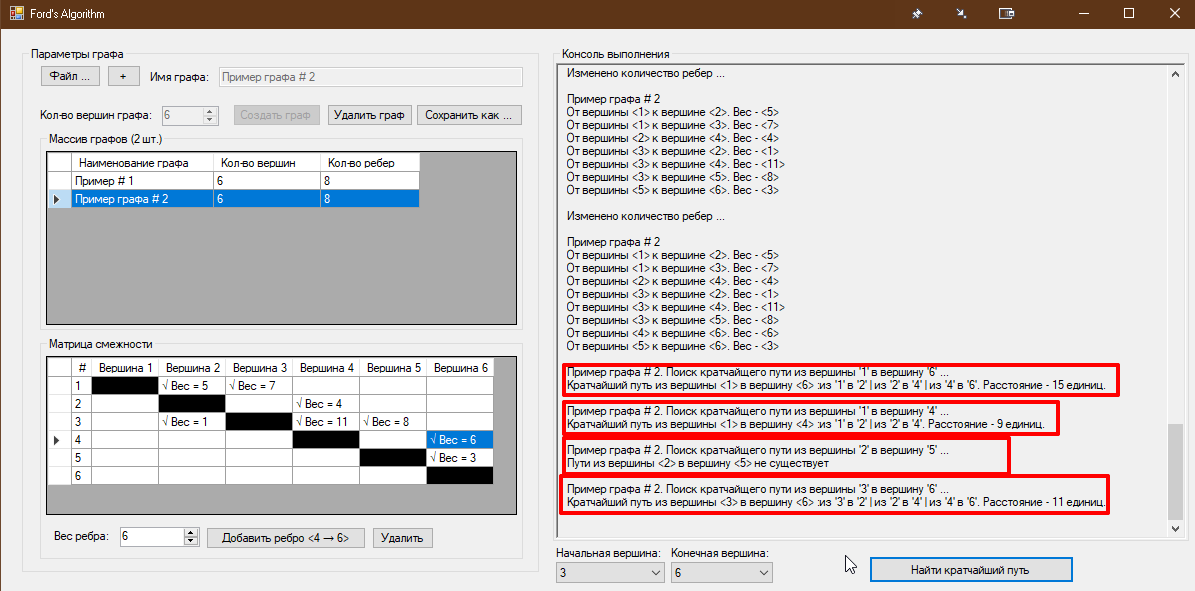


Рисунок 3.6 – Результат расчетов на графе Пример №2

Пример №3

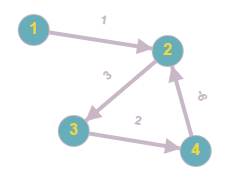


Рисунок 3.7 – Эскиз графа - Пример №3

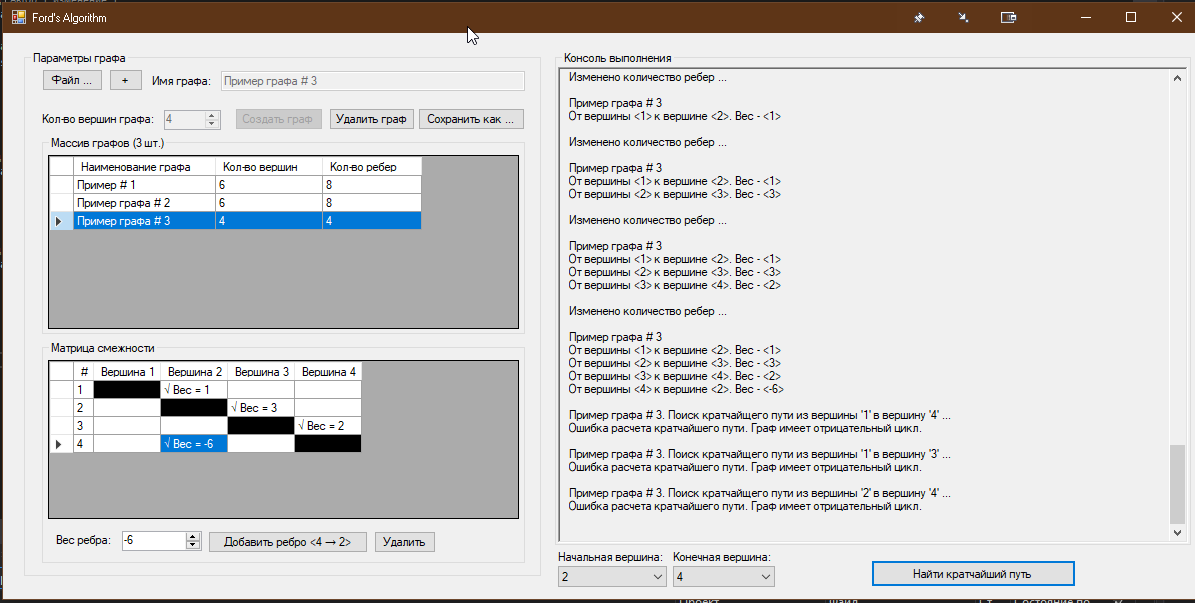
С помощью таблицы матрицы смежности составим граф (См. рисунок 3.8).

Рисунок 3.8 – Граф – Пример №3

Определим кратчайший путь из вершины 1 к вершине 4

Определим кратчайший путь из вершины 1 к вершине 3

Определим кратчайший путь из вершины 2 к вершине 4

Рассчитать кратчайший путь на данном графе невозможно, так как имеется отрицательный цикл.

Результат расчетов представлен на Рисунке 3.9.

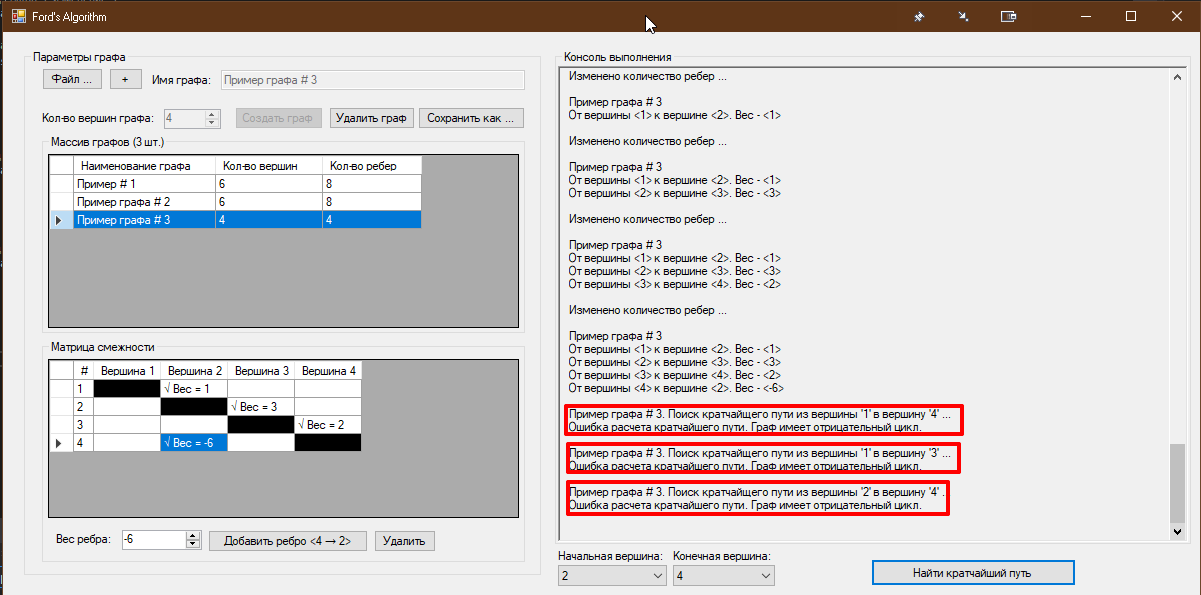


Рисунок 3.9 – Результат расчетов на графе Пример №3

**ЗАКЛЮЧЕНИЕ**

В рамках выполнения данной учебной практики было разработано приложение на языке C#, обеспечивающее поиск кратчайшего пути на графах по алгоритму Форда – Беллмана. Изучая литературу по теории графов, было обнаружено, что графы очень часто применяются в программировании, так как в виде графа можно представить почти любую систему объектов. При разработке приложения возникла трудность в изображении интерфейса. Но вскоре было принято решение реализовать отдельный элемент управления, который в будущем можно встроить в любую экранную форму. Так же, я более подробно узнал, как на языке C# реализуются прочие алгоритмы, такие как как алгоритм Дейкстры, алгоритм обхода в ширину и т.д.

**СПИСОК ЛИТЕРАТЕРЫ**

1. М.Ю. Звягин, М.С. Беспалов, А.В. Александров. Прикладные алгоритмы на графах. Учебное пособие. Стр. 44.

**ПРИЛОЖЕНИЕ А**

Вершина

[Serializable]

public class Вершина

{

public int НомерВершины;

public int? Потенциал;

public Вершина Источник;

public Вершина(int НомерВершины)

{

this.НомерВершины = НомерВершины;

}

}

Ребро

[Serializable]

public class Ребро

{

public int Вес;

public Вершина ВекторНачало;

public Вершина ВекторКонец;

public Ребро(int Вес, Вершина ВекторНачало, Вершина ВекторКонец)

{

this.Вес = Вес;

this.ВекторНачало = ВекторНачало;

this.ВекторКонец = ВекторКонец;

}

}

Граф

[Serializable]

public class Граф

{

public string ИмяГрафа;

[NonSerialized]public Guid? UIDГрафа;

public List<Вершина> Вершины = new List<Вершина>();

public List<Ребро> Ребра = new List<Ребро>();

public Граф(string ИмяГрафа, int КоличествоВершин = 0)

{

this.UIDГрафа = Guid.NewGuid();

this.ИмяГрафа = ИмяГрафа;

for (var i = 1; i <= КоличествоВершин; i++)

{

ДобавитьВершину(new Вершина(i));

}

}

public Граф(Guid UIDГрафа, string ИмяГрафа, int КоличествоВершин = 0)

{

this.UIDГрафа = UIDГрафа;

this.ИмяГрафа = ИмяГрафа;

for (var i = 1; i <= КоличествоВершин; i++)

{

ДобавитьВершину(new Вершина(i));

}

}

public int КоличествоВершин

{

get

{

return Вершины.Count();

}

}

public int КоличествоРебер

{

get

{

return Ребра.Count();

}

}

public void ДобавитьВершину(Вершина вершина)

{

if (Вершины.Where(f => f.НомерВершины == вершина.НомерВершины).Any())

{

MessageBox.Show($"Вершина c номером '{вершина.НомерВершины}' уже существует в графе.",

"", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

Вершины.Add(вершина);

}

public void ДобавитьРебро(Ребро ребро)

{

if (Ребра.Where(f => f.ВекторКонец.НомерВершины == ребро.ВекторКонец.НомерВершины && f.ВекторНачало.НомерВершины == ребро.ВекторНачало.НомерВершины).Any())

{

if(MessageBox.Show($@"Ребро c вершинами '{ребро.ВекторНачало.НомерВершины} → {ребро.ВекторКонец.НомерВершины}' уже существует в графе.

Удалить его?",

"", MessageBoxButtons.YesNo, MessageBoxIcon.Question) != DialogResult.Yes)

{

return;

}

else

{

УдалитьРебро(Ребра.Where(f => f.ВекторКонец.НомерВершины == ребро.ВекторКонец.НомерВершины && f.ВекторНачало.НомерВершины == ребро.ВекторНачало.НомерВершины).Single());

return;

}

}

if (ребро.ВекторНачало.НомерВершины == ребро.ВекторКонец.НомерВершины)

{

MessageBox.Show($"Начальная и конечная вершина добавляемого ребра, имеет один и тот же номер.",

"", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

Ребра.Add(ребро);

}

public void УдалитьРебро(Ребро ребро)

{

if (Ребра.Where(f => f.Equals(ребро)).Any())

{

Ребра.Remove(ребро);

}

}

public Вершина ВершинаПоНомеру(int НомерВершины)

{

var вершина = Вершины.Where(f => f.НомерВершины == НомерВершины).Single();

return вершина;

}

public void ОчиститьГраф()

{

Вершины.Clear();

Ребра.Clear();

}

public string ОпределитьКороткийПуть(int ПутьСтарт, int ПутьФиниш)

{

if (Вершины.SingleOrDefault(f => f.НомерВершины == ПутьСтарт) == null)

{

MessageBox.Show(

$"Вершина под номером <{ПутьСтарт}> не найдена.",

"", MessageBoxButtons.OK, MessageBoxIcon.Error);

return $"Вершина под номером <{ПутьСтарт}> не найдена.";

}

if (Вершины.SingleOrDefault(f => f.НомерВершины == ПутьФиниш) == null)

{

MessageBox.Show(

$"Вершина под номером <{ПутьФиниш}> не найдена.",

"", MessageBoxButtons.OK, MessageBoxIcon.Error);

return $"Вершина под номером <{ПутьФиниш}> не найдена.";

}

var INF = ОпределитьБесконечность(); ///значение бесконечности

Вершины.ForEach(f => f.Источник = null);

Вершины.ForEach(f => f.Потенциал = INF);

Вершины.Single(f => f.НомерВершины == ПутьСтарт).Потенциал = 0;

for (var i = 1; i < Вершины.Count(); i++)

{

РасчетПотенциалаВершин(Вершины.Single(f => f.НомерВершины == ПутьСтарт), ref i);

}

foreach (var ребро in Ребра)

{

if (ребро.ВекторКонец.Потенциал > (ребро.ВекторНачало.Потенциал + ребро.Вес))

{

MessageBox.Show(

"Ошибка расчета кратчайшего пути. Граф имеет отрицательный цикл.",

"", MessageBoxButtons.OK, MessageBoxIcon.Error);

return "Ошибка расчета кратчайшего пути. Граф имеет отрицательный цикл.";

}

}

var КратчайшийПутьlist = new List<string>();

КратчайшийПуть(ВершинаПоНомеру(ПутьФиниш), ref КратчайшийПутьlist);

КратчайшийПутьlist.Reverse();

if (!КратчайшийПутьlist.Any())

{

return $"Пути из вершины <{ПутьСтарт}> в вершину <{ПутьФиниш}> не существует";

}

return $"Кратчайший путь из вершины <{ПутьСтарт}> в вершину <{ПутьФиниш}> :{String.Join(" | ", КратчайшийПутьlist)}. Расстояние - {ВершинаПоНомеру(ПутьФиниш).Потенциал} единиц.";

}

public DataTable ПостороитьМатрицуСмежности()

{

var Матрица = new DataTable();

Матрица.Columns.Add(new DataColumn() { ColumnName = "#" });

for (var i = 0; i<Вершины.Count; i++ )

{

Матрица.Columns.Add(new DataColumn() { ColumnName = $"Вершина {i + 1}", DataType = typeof(string) });

}

for (var i = 0; i < Вершины.Count; i++)

{

Матрица.Rows.Add(Матрица.NewRow());

}

for(var row = 1; row <= Матрица.Rows.Count; row++)

{

Матрица.Rows[row - 1][0] = row;

for (var col = 1; col <= Матрица.Rows.Count; col++)

{

var ребро = Ребра.Where(f => f.ВекторНачало.НомерВершины == row && f.ВекторКонец.НомерВершины == col).SingleOrDefault();

if (ребро != null)

{

Матрица.Rows[row - 1][col] = $"√ Вес = {ребро.Вес}";

}

}

}

return Матрица;

}

private List<Вершина> РасчетПотенциалаВершин(Вершина вершина, ref int УровеньДостижимости)

{

var СмежныеВершины = new List<Вершина>();

var Счетчик = УровеньДостижимости ;

Счетчик--;

foreach(var ребро in Ребра.Where(f => f.ВекторНачало.Equals(вершина)).ToList())

{

if (ребро.ВекторКонец.Потенциал > (ребро.ВекторНачало.Потенциал + ребро.Вес))

{

ребро.ВекторКонец.Потенциал = (ребро.ВекторНачало.Потенциал + ребро.Вес);

ребро.ВекторКонец.Источник = ребро.ВекторНачало;

}

if (Счетчик > 0)

{

СмежныеВершины.AddRange(РасчетПотенциалаВершин(ребро.ВекторКонец, ref Счетчик));

}

else

{

СмежныеВершины.Add(ребро.ВекторКонец);

}

}

return СмежныеВершины.Distinct().ToList();

}

private void КратчайшийПуть(Вершина вершина, ref List<string> l)

{

if (вершина.Источник != null)

{

l.Add($"из '{вершина.Источник.НомерВершины}' в '{вершина.НомерВершины}'");

КратчайшийПуть(вершина.Источник, ref l);

}

}

private int? ОпределитьБесконечность()

{

if (!Вершины.Any())

{

MessageBox.Show("Граф не содержит вершин", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

return null;

}

if (!Ребра.Any())

{

MessageBox.Show("Граф не содержит ребер", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

return null;

}

return Ребра.Sum(f => f.Вес) + 1;

}

}

**ПРИЛОЖЕНИЕ Б**

public partial class UserControlНовыйГраф : UserControl

{

private class МассивГрафовClass : DataTable

{

public МассивГрафовClass()

{

this.Columns.Add("UIDГрафа", typeof(Guid));

this.Columns.Add("Наименование графа", typeof(string));

this.Columns.Add("Кол-во вершин", typeof(int));

this.Columns.Add("Кол-во ребер", typeof(int));

this.PrimaryKey = new DataColumn[] { this.Columns["UIDГрафа"] };

}

public void ДобавитьСтроку(Guid UIDГрафа, string НаименованиеГрафа, int КоличествоВершин = 0, int КоличествоРебер = 0)

{

var nRow = this.NewRow();

nRow["UIDГрафа"] = UIDГрафа;

nRow["Наименование графа"] = НаименованиеГрафа;

nRow["Кол-во вершин"] = КоличествоВершин;

nRow["Кол-во ребер"] = КоличествоРебер;

this.Rows.Add(nRow);

this.AcceptChanges();

}

public void ОбновитьСтроку(Guid UIDГрафа, string НаименованиеГрафа, int КоличествоВершин = 0, int КоличествоРебер = 0)

{

var \_Row = this.Rows.Find(UIDГрафа);

\_Row["Наименование графа"] = НаименованиеГрафа;

\_Row["Кол-во вершин"] = КоличествоВершин;

\_Row["Кол-во ребер"] = КоличествоРебер;

\_Row.AcceptChanges();

}

public void УдалитьСтроку(Guid? UIDГрафа)

{

if (UIDГрафа!= null)

{

this.Rows.Remove(this.Rows.Find(UIDГрафа));

this.AcceptChanges();

}

}

public int ИндексСтрокиПоUIDграфа(Guid UIDГрафа)

{

for(var i = 0; i < Rows.Count; i++)

{

if (((Guid)Rows[i]["UIDГрафа"]).Equals(UIDГрафа))

{

return i;

}

}

return 0;

}

}

private МассивГрафовClass МассивГрафовDT = new МассивГрафовClass();

private List<Граф> МассивГрафовList = new List<Граф>();

private int МатрицаТекущийСтолбец;

private int МатрицаТекущаяСтрока;

public Guid? UIDТекущегоГрафа

{

get

{

var curRow = gridViewМассивГрафов.SelectedRows.Cast<DataGridViewRow>().SingleOrDefault();

if (curRow != null)

{

return (Guid)curRow.Cells["UIDГрафа"].Value;

}

return null;

}

}

public Граф ТекущийГраф

{

get

{

return МассивГрафовList.SingleOrDefault(f => f.UIDГрафа.Equals(UIDТекущегоГрафа));

}

}

public UserControlНовыйГраф()

{

InitializeComponent();

gridViewМассивГрафов.DataSource = МассивГрафовDT;

gridViewМассивГрафов.Columns["UIDГрафа"].Visible = false;

}

public event EventHandler КоличествоРеберChanged;

public event EventHandler МассивГрафовSelectionChanged;

public event EventHandler МассивГрафовДобавление;

public event EventHandler МассивГрафовУдаление;

private void buttonНовыйГраф\_Click(object sender, EventArgs e)

{

textBoxИмяГрафа.Enabled = true;

numericКоличествоВершин.Enabled = true;

buttonСоздатьГраф.Enabled = true;

buttonУдалитьРебро.Enabled = true;

numericКоличествоВершин.Value = 2;

textBoxИмяГрафа.Text = $"Новый граф # {МассивГрафовList.Count + 1}";

textBoxИмяГрафа.Focus();

textBoxИмяГрафа.SelectAll();

}

private void buttonСоздатьГраф\_Click(object sender, EventArgs e)

{

var UID = Guid.NewGuid();

МассивГрафовList.Add(new Граф(UID, textBoxИмяГрафа.Text, (int)numericКоличествоВершин.Value));

МассивГрафовDT.ДобавитьСтроку(UID, textBoxИмяГрафа.Text, (int)numericКоличествоВершин.Value);

МассивГрафовДобавление?.Invoke(this, new EventArgs());

textBoxИмяГрафа.Enabled = false;

numericКоличествоВершин.Enabled = false;

buttonСоздатьГраф.Enabled = false;

gridViewМассивГрафов.Rows[МассивГрафовDT.ИндексСтрокиПоUIDграфа(UID)].Selected = true;

groupBox3.Text = $"Массив графов ({gridViewМассивГрафов.RowCount} шт.)";

}

private void buttonУдалитьГраф\_Click(object sender, EventArgs e)

{

МассивГрафовList.Remove(МассивГрафовList.SingleOrDefault(f=>f.UIDГрафа.Equals(UIDТекущегоГрафа)));

МассивГрафовDT.УдалитьСтроку(UIDТекущегоГрафа);

groupBox3.Text = $"Массив графов ({gridViewМассивГрафов.RowCount} шт.)";

gridViewМатрицаСмежности.DataSource = (ТекущийГраф == null) ? null : ТекущийГраф.ПостороитьМатрицуСмежности();

МассивГрафовУдаление?.Invoke(this, new EventArgs());

}

private void buttonДобавитьРебро\_Click(object sender, EventArgs e)

{

var curCell = gridViewМатрицаСмежности.CurrentCell;

if (curCell == null) { return; }

МассивГрафовList.Single(f => f.UIDГрафа.Equals(UIDТекущегоГрафа))?.ДобавитьРебро

(

new Ребро

(

(int)numericВесРебра.Value,

ТекущийГраф.ВершинаПоНомеру(curCell.RowIndex + 1),

ТекущийГраф.ВершинаПоНомеру(curCell.ColumnIndex)

)

);

МассивГрафовDT.ОбновитьСтроку(ТекущийГраф.UIDГрафа.Value, ТекущийГраф.ИмяГрафа, ТекущийГраф.КоличествоВершин, ТекущийГраф.КоличествоРебер);

gridViewМатрицаСмежности.DataSource = ТекущийГраф.ПостороитьМатрицуСмежности();

gridViewМатрицаСмежности[МатрицаТекущийСтолбец, МатрицаТекущаяСтрока].Selected = true;

КоличествоРеберChanged?.Invoke(this, new EventArgs());

}

private void buttonУдалитьРебро\_Click(object sender, EventArgs e)

{

var curCell = gridViewМатрицаСмежности.CurrentCell;

if (curCell == null) { return; }

var реброУделяемое = ТекущийГраф.Ребра.Where

(

f => f.ВекторНачало.НомерВершины == (curCell.RowIndex + 1) &&

f.ВекторКонец.НомерВершины == curCell.ColumnIndex

).SingleOrDefault();

МассивГрафовList.Single(f => f.UIDГрафа.Equals(UIDТекущегоГрафа))?.УдалитьРебро(реброУделяемое);

МассивГрафовDT.ОбновитьСтроку(ТекущийГраф.UIDГрафа.Value, ТекущийГраф.ИмяГрафа, ТекущийГраф.КоличествоВершин, ТекущийГраф.КоличествоРебер);

gridViewМатрицаСмежности.DataSource = ТекущийГраф.ПостороитьМатрицуСмежности();

gridViewМатрицаСмежности[МатрицаТекущийСтолбец, МатрицаТекущаяСтрока].Selected = true;

}

private void buttonСохранитьКак\_Click(object sender, EventArgs e)

{

if (gridViewМассивГрафов.SelectedRows.Cast<DataGridViewRow>().SingleOrDefault() == null)

{

return;

}

var saveFileDialog = new SaveFileDialog()

{

Filter = "Файл графа (\*.graf)|\*.graf",

FilterIndex = 1,

RestoreDirectory = true,

FileName = $"{ТекущийГраф.ИмяГрафа}.graf"

};

if (saveFileDialog.ShowDialog() != DialogResult.OK)

{

return;

}

var stream = File.Create(saveFileDialog.FileName);

var formatter = new BinaryFormatter();

formatter.Serialize(stream, ТекущийГраф);

stream.Close();

}

private void buttonОткрытьГраф\_Click(object sender, EventArgs e)

{

var openFileDialog = new OpenFileDialog()

{

Filter = "Файл графа (\*.graf)|\*.graf",

FilterIndex = 1,

RestoreDirectory = true,

Multiselect = false

};

if(openFileDialog.ShowDialog() != DialogResult.OK)

{

return;

}

var stream = File.OpenRead(openFileDialog.FileName);

var formatter = new BinaryFormatter();

var openГраф = formatter.Deserialize(stream) as Граф;

openГраф.UIDГрафа = Guid.NewGuid();

МассивГрафовList.Add(openГраф);

МассивГрафовDT.ДобавитьСтроку(openГраф.UIDГрафа.Value, openГраф.ИмяГрафа, openГраф.КоличествоВершин, openГраф.КоличествоРебер);

gridViewМассивГрафов.Rows[МассивГрафовDT.ИндексСтрокиПоUIDграфа(openГраф.UIDГрафа.Value)].Selected = true;

}

private void gridViewМассивГрафов\_SelectionChanged(object sender, EventArgs e)

{

var focusDR = gridViewМассивГрафов.SelectedRows.Cast<DataGridViewRow>().SingleOrDefault();

if(focusDR != null)

{

gridViewМатрицаСмежности.DataSource = ТекущийГраф.ПостороитьМатрицуСмежности();

}

МассивГрафовSelectionChanged?.Invoke(this, new EventArgs());

}

private void gridViewМатрицаСмежности\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if ((e.ColumnIndex == 0) || ((e.RowIndex + 1) == (e.ColumnIndex)))

{

panel2.Enabled = false;

buttonДобавитьРебро.Text = $"Добавить ребро";

}

else

{

panel2.Enabled = true;

buttonДобавитьРебро.Text = $"Добавить ребро <{e.RowIndex + 1} → {e.ColumnIndex}>";

}

МатрицаТекущаяСтрока = e.RowIndex;

МатрицаТекущийСтолбец = e.ColumnIndex;

}

private void gridViewМатрицаСмежности\_ColumnAdded(object sender, DataGridViewColumnEventArgs e)

{

e.Column.SortMode = DataGridViewColumnSortMode.NotSortable;

if (e.Column.Name.Equals("#"))

{

e.Column.ReadOnly = true;

}

}

private void gridViewМатрицаСмежности\_DataSourceChanged(object sender, EventArgs e)

{

foreach(DataGridViewColumn col in gridViewМатрицаСмежности.Columns)

{

foreach(DataGridViewRow row in gridViewМатрицаСмежности.Rows)

{

if ((row.Index + 1) == (col.Index))

{

gridViewМатрицаСмежности[col.Index, row.Index].Style.BackColor = System.Drawing.Color.Black;

}

}

}

}

}