МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Системное программирование и компьютерные технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему: «Применение алгоритмов интеллектуального анализа данных в системе мониторинга и контроля сетевого оборудования»

Студент И.С. Кучерявый
(И.О. Фамилия) (личная подпись)

Руководитель доцент, Н.А. Сосина
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Аннотация

Название дипломной работы: «Венгерский метод решения задачи о назначениях».

Данная дипломная работа состоит из 83 страниц, 23 рисунков, 1 приложения; использованных источников – 22.

Цель работы - изучить принцип работы венгерского алгоритма, обосновать его эффективность, осуществить программную реализацию и визуализацию данного алгоритма.

Предметом исследования моей выпускной квалификационной работы является задача об оптимальном назначение, а объектом исследования - алгоритм венгерского метода решения данной задачи.

В работе были выполнены следующие задачи:

1. Исследован и изучен принцип работы алгоритма венгерского метода.

2. Исследованы другие альтернативные алгоритмы решения задачи об оптимальном назначении.

3. Разработан программный код, который решает задачу оптимального назначения используя венгерский метод.

4. Предоставлен графический интерфейс созданной программы, которая решает задачу оптимального назначения, используя венгерский метод и демонстрирует все промежуточные этапы выполнения алгоритма.

5. Проверена эффективность данной программы для различных входных данных и ее соответствие с венгерским методом.

Программа может быть применена с целью решения задачи о назначениях для любой матрицы стоимостей.

ABSTRACT

The topic of the given bachelor's thesis is "Hungarian method of the assignment problem solving".

The bachelor's thesis consists of an explanatory note on 83 pages, including 23 figures, the list of 21 references, and 1 appendix.

The purpose of this work is to study the principle of operation of the Hungarian algorithm, to justify its effectiveness, to provide software implementation and to visualize this algorithm.

The subject of the research is the problem of optimal assignment, and the object of the research is the algorithm of the Hungarian method for solving this problem.

The following tasks were completed:

1. to investigate and study the principle of operation of the algorithm of the Hungarian method,

2. to investigate other alternative algorithms for solving the optimal assignment problem,

3. to develop a program code that solves the optimal assignment problem using the Hungarian method,

4. to provide the graphical interface of the created program, which solves the optimal assignment problem using the Hungarian method and demonstrates all the intermediate stages of the algorithm execution,

5. to verify the effectiveness of this program for various input data and its compliance with the Hungarian method.

The program can be used to solve the assignment problem for any cost matrix.

Содержание

ВВЕДЕНИЕ

Важной проблемой современной теории управления является оптимизация. Решение данной задачи нуждается в разработке и практическом применении методов оптимизации, базирующихся на использовании современных ЭВМ. Между прикладными задачами оптимизации особое место занимают задачи дискретного программирования. Их делят на задачи комбинаторного типа, допустимое множество которых имеет конечное количество точек и задачи целочисленного программирования, где переменные принимают целочисленные значения и задачи частично дискретного программирования, в которых лишь часть переменных принимает дискретные значения.

Для своей выпускной квалификационной работы я выбрал тему «Венгерский метод решения задачи о назначениях». Текущая задача является типичным примером задач дискретного программирования. В работе осуществляется визуализация задачи.

Предметом исследования моей ВКР является задача об оптимальном назначение, а объектом исследования - алгоритм венгерского метода решения данной задачи.

Цель исследования - изучить принцип работы венгерского алгоритма, обосновать его эффективность, осуществить программную реализацию и визуализацию данного алгоритма.

Согласно с целью исследования моей ВКР ставятся следующие задачи:

1. Исследовать и изучить принцип работы алгоритма венгерского метода.
2. Исследовать другие альтернативные алгоритмы решения задачи об оптимальном назначении.

3. Разработать программный код, который будет решать задачу оптимального назначения используя венгерский метод.

4. Предоставить графический интерфейс, предварительно написать код, создать программу, которая решает задачу оптимального назначения, используя венгерский метод и демонстрирует все промежуточные этапы выполнения алгоритма.

5. Проверить эффективность данной программы для различных входных данных и ее соответствие с венгерским методом.

Для комплексного изучения предмета ВКР проведены следующие исследования:

· Сравнение нескольких различных алгоритмов, и их анализ;

· Эксперимент - реализация алгоритма венгерского метода и проверка его эффективности на практике;

· Формализация - отражение принципа действия венгерского метода в знаковой форме языка программирования Delphi ;

· Анализ алгоритма, и его отдельных этапов выполнения.

Научная новизна исследования заключается в разработке программы, которая позволяет визуально проследить поэтапную работу венгерского метода.

1. ПОСТАНОВКА И РЕШЕНИЕ ЗАДАЧИ ОБ ОПТИМАЛЬНОМ НАЗНАЧЕНИИ

1.1. Условие задачи оптимального назначения. Алгоритмы решения

Задача об оптимальном назначении является одной из основных типовых задач дискретного программирования. Дискретное программирование - раздел программирования, который предусматривает работу с дискретными (целочисленными) данными. Также основными задачами дискретного программирования считают задачи о рюкзаке, выбор средств доставки, нахождения кратчайших расстояний между вершинами графа и др.

Для задачи об оптимальном назначении, самой распространенной формулировкой являются следующие условия задачи.

Есть n разновидностей работ и n кандидатов с целью их исполнения. Считается, что каждый из кандидатов $i = 1 \ldots n$ может выполнять любую работу $j = 1 \ldots n$. При этом C_{ij}- расходы, связанные с назначением i-го кандидата на j-й вид работы.

Нужно таким образом распределить кандидатов с целью выполнения работы, чтобы каждый из кандидатов получил единственное назначение, а каждая работа получила единственного исполнителя и суммарные расходы, связанные с назначением, были минимальными.

Для решения данной задачи существуют различные методы и алгоритмы, в частности:

1. Алгоритм с использованием элементов комбинаторики.
2. Алгоритм метода– Мака.
3. Алгоритм венгерского метода.

1.2. Алгоритм с использованием элементов комбинаторики

Комбинаторика - это раздел математики, предназначенный для решения задач выбора и места расположения элементов некоторого, как правило конечного, множества в согласовании с данными правилами. Каждое правило назначает метод построения некой конструкции из элементов первоначального множества, носящей называние комбинаторной конфигурацией.

«Комбинаторика связана с такими разделами математики, как теория множеств, теория графов, теория вероятностей, теория чисел и др. Комбинаторные методы применяются в теории случайных процессов, статистике, математическом программировании, вычислительной математике, планировании экспериментов» [22, с. 63].

Как простой пример комбинаторных конфигураций, существуют такие операции как: перестановка, размещение, комбинация.

Перестановка - перемена расположения элементов множества с целью формирования новейшей комбинации элементов. Из числа n элементов, позволено создать n! всевозможных комбинаций.

Размещение - создание множеств размерностью в m элементов из множества с количеством элементов n. Если множество m состоит из различных, неповторяющихся элементов, то общее количество множеств размерностью m, которые будут отличаться между собой по расположению элементов, вычисляется по формуле:

$$A_n^m = \frac{n!}{(n-m)!}$$

Если же условие задачи предполагает повторяемость элементов, то с n элементов можно составить значительно большее количество множеств, содержащие m элементов. Тогда общее количество множеств размерностью m исчисляться по формуле [11, с. 32]:

$$A_n^m = n^m$$

Комбинация - способ выбора нескольких вещей из большей группы, где (в отличие от размещения) порядок не имеет значения. Количество возможных

комбинаций размерностью m из n элементов вычисляется по формуле [12, с. 167-168]:

$$C_n^m = \frac{n!}{m!(n-m)!}$$

Задача оптимального назначения работников решается с применением метода перестановки.

Например, если в задаче задано 4 работника, то мы должны составить все возможные комбинации из чисел 1, 2, 3, 4. Комбинация 3124 говорит о том, что 3-го работника следует назначить на 1 (положение цифры 3 в комбинации) работу, 1-го работника на 2 работу, 2-го работника на 3 и 4-го на 4 работу. Аналогичным образом происходят назначения работников на определенный вид работы в соответствии с другими комбинаций. Для каждой из комбинаций следует вычислить общую сумму затрат и в конечном итоге выбрать ту, для которой сумма расходов минимальна.

Пошаговый алгоритм с учетом принципа формирования комбинаций такой:

1. Вводим число n - это количество работ и работников.

2. Генерируем последовательность A (i) = 1 2 3 ... n - распределение работников на работы от 1 до n.

3. Находим сумму затрат для данной последовательности.

4. Идем с правого конца последовательности A, пока не найдем цифру, меньшую, чем предыдущая. Если такая цифра найдена, то переходим к пункту 5. Если нет - к пункту 11.

5. Среди всех цифр, которые находятся справа от найденной, ищем такую, которая является самой большой и её величина больше найденной. Меняем ее местами с найденной.

6. Все цифры, находящиеся справа от позиции найденной цифры, зеркально отражаем.

7. Находим общую сумму затрат для данной последовательности.

8. Переходим к пункту 4.

9. Сравниваем все найденные суммы и находим наименьшую.

10. Выводим наименьшую сумму затрат, а также информацию о том, какого работника на какую работу следует назначить (согласно комбинации, которая соответствует наименьшей сумме).

11. Конец.

1.3. Алгоритм метода Мака

Венгерский метод (его описание представлено ниже) базируется на некоторых достаточно непростых комбинаторных свойствах матрицы. Его достаточно сложно программировать. Метод Мака, в сравнении с остальными, обладает преимущество несложного интуитивного обоснования. Это - логический процесс. Данный метод основывается на идее взятия наименьшего элемента в каждой строке. Наименьшие элементы строк не распределены в каждом n столбце матрицы. В данном случае используется идея прибавления (или вычитания) эквивалентного значения для каждого элемента строки или столбца, с целью расположения минимальных элементов строк по столбцам.

1. Обозначаем минимальный элемент каждой строки матрицы затрат значком *. Если таких элементов несколько, обозначаем любой из них.

2. Для каждой строки, что имеет другой минимальный элемент, просматриваем столбец, к которому этот элемент принадлежит. Возможны случаи:

а) столбец не имеет обозначенных элементов;

б) столбец имеет, по крайней мере, один обозначенный элемент.

3. В случае с а) обозначаем другой минимальный элемент строки значком *. Все остальные отметки в этой строке ликвидируются. В случае с б) обозначаем другой минимальный элемент строки отметкой ^, если элемент этой строки со значком * не является единственным обозначенным элементом в своем столбце.

4. Действия пунктов 2 и 3 повторяем последовательно для всех строк, имеющих более одного минимального элемента.

5. Если каждый столбец матрицы расходов имеет элемент с пометкой *, то алгоритм заканчивается: эти элементы определяют оптимальные назначения. Иначе переходим к следующему пункту.

6. Обозначаем (символом &) столбцы, имеющие более одного обозначенного элемента. Они образуют множество B, остальные столбцы матрицы затрат образуют множество A.

7. Для каждой строки матрицы затрат, в которой элемент с пометкой * принадлежит множеству B, находится минимальная разница между элементами множества A и элементом с отметкой *.

8. Находим наименьшую из указанных разниц, добавляем ее к каждому элементу множества и возвращаемся к пункту 2.

1.4. Алгоритм венгерского метода

Венгерский метод является одним из самых интересных и самых распространенных методов решения задач о назначении. Главную идею данного метода впервые высказал венгерский математик Э.Эгервари (отсюда и название метода) задолго до возникновения теории линейного программирования [14, с. 211].

Сам алгоритм был разработан и опубликован Гарольдом Куном в 1955 году. В 1957 году Джеймс Манкрес показал, что этот алгоритм работает за четкое полиномиальное время (то есть, за время порядка полинома от *n*-ой степени). Таким образом в литературе этот алгоритм стал известнее не только как «венгерский», но и как «алгоритм Куна-Манкреса» или «алгоритм Манкреса» [13, с. 18].

Непосредственно алгоритм решения задачи состоит из следующих шагов:

1. Вычитаем в матрице С от каждого элемента *i*-й строки минимальный элемент этой строки, $i = 1 \ldots n$.

2. Вычитаем от каждого элемента *j*-го столбца преобразованной матрицы затрат его минимальный элемент, $j = 1 \ldots n$. В результате выполнения двух пунктов каждая строка и каждый столбец матрицы расходов имеют, по крайней мере, один 0.

3. Просматриваем последовательно строки матрицы затрат, начиная с первой. Если строка имеет только один необозначенный 0, обозначаем его символом * и зачеркиваем (с помощью символа ^) все нули в этом же столбце. Ноль считается обозначен, если он обозначен символом *. Повторяем эти действия, пока каждая строка не останется без необозначенных нулей, или будет иметь их по крайней мере 2.

4. Действия пункта 3 повторяем для всех столбцов матрицы затрат.

5. Действия пунктов 3 и 4 повторяем последовательно (если необходимо), пока не получим один из трех возможных случаев:

а) каждая строка имеет назначение (с пометок 0 *)

б) есть по крайней мере два необозначенных нуля в некоторых строках и некоторых столбцах матрицы затрат;

в) нет обозначенных нулей и полное назначение еще не получено (число нулей с пометкой * меньше n).

6. В случае а) задача об оптимальных назначении решена: x_{ij} *, соответствующие 0 * равны 1. Остальные - 0, конец алгоритма.

В случае б) произвольно выбираем один из необозначенных нулей, обозначаем его символом *, зачеркиваем остальные нули в той же строке и в том же столбце и возвращаемся к пункту 3. Если имеет место случай в), то переходим к пункту 7.

7. Обозначаем символом # строки, для которых не получено назначение (в которых нет 0 *). Такие строки считаем обозначенными, остальные – не назначенными. Такую же терминологию будем использовать и для столбцов матрицы затрат.

8. Обозначаем символом # еще необозначенные столбцы, которые имеют зачеркнутый 0 (обозначен символом ^) в обозначенных строках.

9. Обозначаем символом # еще необозначенные строки, которые имеют назначение (то есть 0 *) в обозначенных столбцах.

10. Повторяем действия пунктов 8 и 9 до тех пор, пока больше не сможем обозначать строки и столбцы матрицы затрат.

11. Зачеркиваем (с помощью отметки &) необозначенные строки и обозначенные столбцы матрицы затрат.

12. Находим минимальный не зачеркнутый элемент матрицы затрат, отнимаем его от элементов каждого из не зачеркнутых строк, добавляем к элементам всех зачеркнутых столбцов и переходим к пункту 3. При этом отметки элементов матрицы затрат (* и ^) теряют свою силу.

Пример 1. Решить задачу оптимального назначения с матрицей затрат C.

Примечание [НА1]: Матрицы набей сам в редакторе формул

$$C = \begin{pmatrix} 3 & 10 & 5 & 9 & 16 \\ 6 & 8 & 11 & 8 & 18 \\ 7 & 13 & 10 & 3 & 4 \\ 5 & 9 & 6 & 21 & 12 \\ 5 & 4 & 11 & 6 & 13 \end{pmatrix} \qquad C_1 = \begin{matrix} & & & & \# \\ \begin{pmatrix} 0^* & 7 & 1 & 6 & 12 \\ 0^\wedge & 2 & 4 & 2 & 11 \\ 4 & 10 & 6 & 0^* & 0^\wedge \\ 0^\wedge & 4 & 0^* & 16 & 6 \\ 1 & 0^* & 6 & 2 & 8 \end{pmatrix} & \begin{matrix} \# \\ \# \\ \\ \\ \end{matrix} \end{matrix}$$

Выполняя действия пунктов 1, 2 алгоритма, которые иногда называют предварительным преобразованиями, получаем эквивалентную матрицу C_1. Обозначаем нули матрицы C_1. в соответствии с пунктами 3, 4, 6 алгоритма. Поскольку количество нулей, обозначенных *, меньше n = 5, то переходим к пункту 7. Выполняя действия пунктов 7-10, обозначаем символом # сначала вторую строчку, затем первый столбец и первую строку матрицы C_1. После выполнения пункта 11 алгоритма получаем матрицу C_2.

$$C_2 = \begin{matrix} & & & & \& \\ \begin{pmatrix} 0^* & 7 & 1 & 6 & 12 \\ 0^\wedge & 2 & 4 & 2 & 11 \\ 4 & 10 & 6 & 0^* & 0^\wedge \\ 0^\wedge & 4 & 0^* & 16 & 6 \\ 1 & 0^* & 6 & 2 & 8 \end{pmatrix} & \begin{matrix} \& \\ \\ \\ \& \\ \& \end{matrix} \end{matrix} \qquad C_3 = \begin{matrix} & \# & & \# & \\ \begin{pmatrix} 0^\wedge & 6 & 0^\wedge & 5 & 11 \\ 0^* & 1 & 3 & 1 & 10 \\ 5 & 10 & 6 & 0^* & 0^\wedge \\ 1 & 4 & 0^* & 16 & 6 \\ 2 & 0^* & 6 & 2 & 8 \end{pmatrix} & \begin{matrix} \# \\ \# \\ \\ \# \\ \end{matrix} \end{matrix}$$

Минимальный не зачеркнутый элемент матрицы C_2 равен 1. После выполнения пункта 12 алгоритма получаем матрицу затрат C_3, в которую также внесены отметки в соответствии с действиями пунктов 3-10 алгоритма. Выполнение пунктов 11, 12 приводит, соответственно, к матрицам C_4 и C_5.

r

$$C_4 = \begin{matrix} & & \& & & \\ \begin{pmatrix} 0^\wedge & 6 & 0^\wedge & 5 & 11 \\ 0^* & 1 & 3 & 1 & 10 \\ 5 & 10 & 6 & 0^* & 0^\wedge \\ 1 & 4 & 0^* & 16 & 6 \\ 2 & 0^* & 6 & 2 & 8 \end{pmatrix} & \begin{matrix} \\ \\ \& \\ \\ \& \end{matrix} \end{matrix} \qquad C_5 = \begin{pmatrix} 0^* & 5 & 0^\wedge & 4 & 10 \\ 0^\wedge & 0^\wedge & 3 & 0^* & 9 \\ 6 & 10 & 7 & 0^\wedge & 0^* \\ 1 & 3 & 0^* & 15 & 5 \\ 3 & 0^* & 7 & 2 & 8 \end{pmatrix}$$

Матрица C_5 содержит также отметки, отражающие действия пунктов 3-5 алгоритма на следующей итерации. Так как количество 0 * равно 5, то получено оптимальное решение X^*:

$$X^* = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

то есть первый работник назначается на выполнение первой работы, второй - четвертой, третьей - пятой, четвертый - третьей, пятый - второй.

1.5. Постановка задачи

Сетевые модели обширно используются в исследовании операций и могут стать массово применены на практике, допустим, ввиду проектирования больших оросительных систем, вычислительных комплексов, транспортных сетей, систем космической связи. С целью решения практических задач, которые связанны со складированием и распределением товаров, календарным планированием, перевозками и также с многими другими задачами были реализованы эффективные алгоритмы, базирующиеся на методах сетевого анализа.

Применение на практике задачи о назначениях весьма широко, так как она представляется задачей управления организацией. В числе данных задач весьма распределено распространение прав, обязанностей, работ, льгот среди членов коллектива, в решении которых принимает участие руководитель, который ответственен за данное распределение [2].

Через отдел подготовки рукописей серьезного издательства проходит большое количество рукописей книг. Данные рукописи нужно разбивать между сотрудниками. Возможно любой рукописи будет дана оценка по следующим критериям как важность, срочность выполнения, тематика. Равным образом, сотрудники могут получить оценки по следующим критериям, как качество работы, индивидуальная «пропускная способность», предпочитаемая тематика и так далее. Нужно распределить рукописи между сотрудниками таким образом, чтобы иметь удовлетворяющее качество выполнения каждой работы при наименьших ресурсных затратах.

Указанные ранее ситуации можно относить к задаче о назначениях, её в свою очередь следует формулировать следующим образом: имеется n исполнителей $i = 1 \dots n$ и имеется n рабочих мест перенумерованных индексом $j = 1 \dots n$. Для всех исполнителей считаются ясными те рабочие места, на которых они могут работать. В случае если исполнитель i назначается на рабочее место j, то его производительность считается известной. $C_{ij}(i, j = \overline{1,n})$ – эффективность

выполнения каждой работы каждым исполнителем. Задача заключается в том, чтобы назначить каждому из исполнителей одну и только одну работу таким образом, чтобы оптимизировать заданную функцию эффективности.

Математическая модель задачи о назначениях выглядит следующим образом [9]:

$F(x) = \sum_{i=1}^{n} \sum_{J=1}^{n} C_{ij} X_{ij} \to min, \qquad \sum_{J=1}^{n} X_{ij} = 1, i = \overline{1,n}, \ \sum_{i=1}^{n} X_{ij} = 1, j = \overline{1,n}, \ X_{ij}\ (0,1), i, j = \overline{1,n}.$ (1.1)

Покажем, данную задачу применяя аппарат теории графов [4]. Допустим дан $2n$ – вершинный двудольный граф $G = (V_1, V_2, E)$, в котором первая доля $V_1 = \{1,2, \dots, i, \dots, n\}$ поставляется во взаимно – однозначное соответствие исполнителям , а вторая доля $V_2 = \{1,2, \dots, j, \dots, n\}$ поставляется во взаимно – однозначное соответствие рабочим местам.

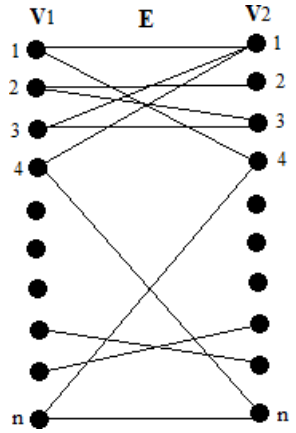

Рисунок 1.1 – $2n$ – вершинный двудольный граф

Если исполнитель i назначен на рабочее место j , то ребро $e = (i, j)$ содержится в E. Каждому ребру $e = (i, j) \in E$ приписан вес $\omega(e) = C_{ij}$. Допустимое решение задачи о назначениях на графе G определяется как совершенное паросочетание $Y = (V_1, V_2, E)$, $E_x \subseteq$E этого графа, $X = \{x\}$ –

множество всех совершенных паросочетаний на G или, что тоже самое множество допустимых значений (МДР)

На МДР X определена целевая функция (ЦФ) $F(x) = \sum_{e \in E_x}^{n} \omega(e) \to min.$

Требуется найти оптимальное решение, а именно, совершенное паросочетание $x^0 = (V_1, V_2, E_{x^0}) \in X$, которое удовлетворяет следующему требованию $F(x^0) = \max_{x \in X} F(x)$.

В главе проанализированы методы решения задачи о назначении. Текущая задача решена с помощью венгерского алгоритма.

2. КОМПЬЮТЕРНАЯ РЕАЛИЗАЦИЯ ВЕНГЕРСКОГО МЕТОДА РЕШЕНИЯ ЗАДАЧИ О НАЗНАЧЕНИЯХ

2.1. Основные принципы венгерского метода

«Задача о назначениях описывает собой индивидуальный случай обычной транспортной задачи и, тем самым, считается задачей транспортного типа. Так как любое вероятное начальное решение задачи о назначении считается вырожденным, симплексный метод будет не эффективным. Задача о назначениях имеет отличительные свойства, которые дали возможность для разработки эффективного метода ее решения, и на данный момент его называют венгерским»

Пусть $C = (c_{ij}) \in M_n(R)$ — матрица стоимости задачи о назначениях (1.1) и $C^* = (c_{ij}^*) \in M_n(R)$ — эквивалентная ей матрица стоимости, т.е. $c_{ij}^* = c_{ij} + d_i + l_j$, $i = \overline{1,n}$, $j = \overline{1,n}$, где d_i, $i = \overline{1,n}$, и l_j, $j = \overline{1,n}$, — некоторые постоянные. Тогда при замене в задаче параметров c_{ij}, входящих в матрицу стоимости C , параметрами c_{ij}^* , $i,j = \overline{1,n}$, из эквивалентной матрицы стоимости C^* оптимальное решение исходной задачи останется неизменным.

В согласии с (1.1), есть возможность утверждать, что для каждого допустимого решения задачи о назначениях матрица переменных $\tilde{X} = (x_{ij}) \in M_n(R)$, содержащая $n(n-1)$ и n единиц, из которых никакие две не относятся к одной и той же строке или одному и тому же столбцу. «Таким образом, если эти n единиц в матрице $\tilde{X}$ расставить в соответствии с расположением элементов системы n независимых нулей эквивалентной матрицы стоимости C^* , то получим допустимое решение рассматриваемой задачи о назначениях. «Более того, найденное допустимое решение является оптимальным решением, так как, ему соответствует нулевое значение целевой функции, определяемой матрицей» C^* , «которое не может быть уменьшено в силу не отрицательности элементов равнозначной матрицы стоимости» C^*

«Алгоритм венгерского метода решения задачи о назначениях состоит из подготовительного этапа и не более чем $n-2$ последовательно повторяющихся

итераций. На подготовительном этапе получают матрицу стоимости C_0, эквивалентная матрице стоимости рассматриваемой задачи о назначениях и содержащую первоначальную систему независимых нулей. На каждой итерации число независимых в преобразованной эквивалентной матрице стоимости не менее чем на единицу Через конечное число итераций система независимых нулей в преобразованной эквивалентной матрице стоимости С0 будет состоять из n элементов, что означает завершение процесса решения рассматриваемой задачи»

Этап подготовки базируется на применении следующих по порядку трех шагов.

Шаг 1. Для всех столбцов матрицы стоимости C задачи о назначениях устанавливают минимальный элемент

$$l_j = \min_{i=\overline{1,n}} c_{ij}\ , j = \overline{1,n}$$

и составляют равнозначную матрицу стоимости $C'_{ij} = (c'_{ij})$, в которой

$$c'_{ij} = c_{ij} - l_j \geq 0, i,j = \overline{1,n}$$

«В результате выполнения первого шага подготовительного этапа получают эквивалентную матрицу стоимости C' , в каждом столбце которой имеется, по крайней мере один нулевой элемент»

Шаг 2. Находят наименьший элемент для каждой строки матрицы стоимости C'

$$d_i = \min_{j=\overline{1,n}} c'_{ij}\ , i = \overline{1,n}$$

и создают равнозначную матрицу стоимости $C_0 = (c^0_{ij})$, в которой

$$c^0_{ij} = c'_{ij} - d_i \geq 0, i,j = \overline{1,n}$$

«В итоге образуется эквивалентная матрица стоимости С0 с неотрицательными элементами, в каждой строке и каждом столбце которой имеется по крайней мере один нулевой элемент»

Шаг «3. В первом столбце матрицы C_0 выбирают любой нулевой элемент и обозначают его как 0^*. Далее во втором столбце выбирают нулевой элемент, не лежащий в одной строке с 0^* первого столбца и тоже обозначают как 0^*. Эту процедуру повторяют для всех последующих столбцов матрицы стоимости C_0 и, таким образом, получается первоначальная система независимых нулей, которая состоит из элементов, обозначенных как 0^*. Эта система не может содержать менее двух элементов»

Легко подтвердить, что у исходной системы независимых нулей не может содержать в себе меньше чем два элемента.

После этого приступим к описанию итерации венгерского алгоритма. Допустим произошло m итераций $(m \geq 0)$, в итоге чего образована равнозначная матрица стоимости C_m.

Шаг 1. «В матрице C_m подсчитываем число элементов в системе независимых нулей, которое обозначим через k. Если $k = n$, то переходим к шагу 2. Если $k < n$, то переходим к шагу 3»

Шаг 2. «В соответствии с системой n независимых нулей эквивалентной матрицы стоимости $C^* = C_m$ в матрице переменных модели $\tilde{X}$ расставляем n единиц, а остальные ее элементы заменяем нулями. Решение завершено»

Шаг 3. «Столбцы матрицы C_m, содержащие 0^*, выделяем знаком „+“, их элементы называются выделенными (остальные элементы матрицы называют невыделенными). Переходим к шагу 4»

Шаг 4. «Если среди невыделенных элементов матрицы C_m есть хотя бы один нуль, то переходим к шагу 5. В противном случае переходим к шагу 9»

Шаг 5. «Если строка, содержащая невыделенный нуль, содержит также и 0^*, то переходим к шагу 6. В противном случае переходим к шагу 7»

Шаг 6. «Найденный невыделенный нуль, обозначаем через $0'$, содержащую его строку отмечаем знаком „+“ и все элементы называем выделенными. Снимаем знак „+“ со столбца, в котором расположен 0^* из выделенной строки, и переходим к шагу 4»

Шаг 7. «Найденный невыделенный нуль, обозначаем через $0'$ и, начиная с него, строим так называемую L – цепочку по следующему правилу: исходный $0'$; далее 0^*, расположенный с ним в одном столбце (если такой найдется); затем $0'$, расположенный в одной строке с предшествующим 0^*; далее 0^* расположенный в одном столбце с предшествующим $0'$(если такой найдется) и т.д. Переходим к шагу 8»

«Действительно, пусть $C_m = (C_{ij}^m) \in M_n(R)$ и существует L – цепочка вида

$$L = (C_{i_1 j_1}^m, C_{i_2 j_1}^m, C_{i_2 j_2}^m, \dots, C_{i_k j_k}^m, C_{i_{k+1} j_k}^m),$$

которая может быть продолжена, причем $C_{i_k j_k}^m$ отмечен как $0'$, $C_{i_{k+1} j_k}^m$ отмечен как 0^*.В этом случае в матрицы C_m столбец с номером j_k, не выделен знаком „+“, так как в этом столбце есть элементы $C_{i_k j_k}^m$ типа $0'$. В то же время этот столбец содержит 0^* (так отмечен элемент $C_{i_{k+1} j_k}^m$). Следовательно, знак выделения столбца с номером j_k был снят и строка с номером i_{k+1} содержит $0'$, так как на шаге 6 она была выделена. Таким образом, L-цепочка может быть продолжена, что противоречит сделанному предположению»

Число элементов в L – цепочке является нечетным. В то же время L– цепочка может состоять из одного элемента, если в одном столбце с рассматриваемым $0'$, нет 0^*.

Шаг 8. «В L – цепочке все 0^* заменяем нулями, а все $0'$ – символами 0^*. В результате чего в эквивалентной матрице стоимости C_m получаем новую систему независимых нулей, число элементов которой на единицу больше числа элементов в предыдущей системе независимых нулей»

Кроме того, вне L – цепочки все $0'$ заменяем на нули и убираем все выделения строк и столбцов C_m. Затем переходим к шагу 1.

Шаг 9. «Среди невыделенных элементов матрицы C_m находим минимальный элемент $\mathrm{h}(\mathrm{h} > 0)$ в силу неотрицательности элементов эквивалентной матрицы стоимости C_m и отсутствия невыделенных нулей). Значение h вычитаем из элементов невыделенных строк и прибавляем к элементам выделенных столбцов. Вновь полученную эквивалентную матрицу стоимости с неотрицательными элементами, в которой по меньшей мере один из невыделенных элементов является нулем, обозначаем через C_m и переходим к шагу 5»

Изобразим являющийся подготовкой этап алгоритма венгерского метода для последующей матрицы стоимости:

$$\mathrm{C} = \begin{pmatrix} 5 & 6 & 7 & 1 \\ 10 & 4 & 6 & 7 \\ 8 & 5 & 3 & 5 \\ 12 & 5 & 9 & 8 \end{pmatrix}$$

Шаг 1. «В первом столбце минимальным является элемент $l_1 = c_{11} = 5$, во втором – $l_2 = c_{22} = 4$, в третьем – $l_3 = c_{33} = 3$, в четвертом – $l_4 = c_{14} = 1$. Вычитаем эти значения из соответствующих столбцов и приходим к матрице»

$$C' = \begin{pmatrix} 0 & 2 & 4 & 0 \\ 5 & 0 & 3 & 6 \\ 3 & 1 & 0 & 4 \\ 7 & 1 & 6 & 7 \end{pmatrix}$$

Шаг 2. «Ищем наименьшие элементы в строках матрицы C''. В первой строке $d_1 = c'_{11} = 0$, во второй – $d_2 = c'_{22} = 0$, вы третьей – $d_3 = c'_{33} = 0$, в четвертой – $d_4 = c'_{42} = 1$. Вычитаем эти значения из соответствующих строк, получаем матрицу»

$$C_0 = \begin{pmatrix} 0 & 2 & 4 & 0 \\ 5 & 0 & 3 & 6 \\ 3 & 1 & 0 & 4 \\ 6 & 0 & 5 & 6 \end{pmatrix}$$

Шаг 3. Отмечаем систему независимых нулей:

$$C_0 = \begin{pmatrix} 0^* & 2 & 4 & 0 \\ 5 & 0^* & 3 & 6 \\ 3 & 1 & 0^* & 4 \\ 6 & 0 & 5 & 6 \end{pmatrix}, \text{ или } C_0' = \begin{pmatrix} 0^* & 2 & 4 & 0 \\ 5 & 0 & 3 & 6 \\ 3 & 1 & 0^* & 4 \\ 6 & 0^* & 5 & 6 \end{pmatrix}$$

«Рассмотрим итерационный процесс венгерского метода решения этой задачи»

«Первая итерация»

Шаг 1. Число k элементов в системе независимых нулей матрицы C_0 равняется трем (пример 1). А потому как $n = 4$ и $k < n$ приступаем к шагу 3.

Шаг 3. Осуществляем выделение столбцов матрицы C_0 , в которых содержится 0^* :

$$C_0 = \begin{matrix} + & + & + & \\ \end{matrix} \begin{pmatrix} 0^* & 2 & 4 & 0 \\ 5 & 0^* & 3 & 6 \\ 3 & 1 & 0^* & 4 \\ 6 & 0 & 5 & 6 \end{pmatrix}$$

Приступаем к шагу 4.

Шаг 4«Один из элементов невыделенного четвертого столбца матрицы C_0 один является нулевым (в первой строке). Переходим к шагу 5»

Шаг 5«В первой строке матрицы C_0, наряду с невыделенным нулем $(c_{14}^0 = 0)$ есть элемент c_0^{11} , выделенный как 0^*. Переходим к шагу 6»

Шаг 6«Найденный невыделенный нуль, обозначаем через $0'$ и выделяем содержащую его первую строку. Снимаем знак выделения с первого столбца, в котором расположен 0^* из первой строки» :

$$C_0 = \begin{matrix} \\ + \end{matrix} \begin{matrix} \quad + & + & \\ \end{matrix} \begin{pmatrix} 0^* & 2 & 4 & 0' \\ 5 & 0^* & 3 & 6 \\ 3 & 1 & 0^* & 4 \\ 6 & 0 & 5 & 6 \end{pmatrix}$$

Приступаем к шагу 4.

Шаг 4 «Среди невыделенных элементов матрицы C_0 нет нулей. Переходим к шагу 9»

Шаг 9 «Среди невыделенных элементов матрицы Со находим минимальный элемент h = min {5, 3, 6, 6, 4, 6} = 3. Значение h = 3 вычитаем от элементов невыделенных строк с номерами 2, 3, 4 и прибавляем к элементам выделенных столбцов с номерами 2, 3. Вновь полученную эквивалентную матрицу стоимости обозначаем через Со» :

$$C_0 = \begin{matrix} \\ + \end{matrix} \begin{matrix} \quad + & + & \\ \end{matrix} \begin{pmatrix} 0^* & 5 & 7 & 0' \\ 2 & 0^* & 3 & 3 \\ 0 & 1 & 0^* & 1 \\ 3 & 0 & 5 & 3 \end{pmatrix}$$

Приступаем к шагу 5.

Шаг 5 «В третьей строке, содержащей невыделенный нуль ($c'_{31} = 0$), есть и выделенный нуль ($c'_{33} = 0^*$). Переходим к шагу 6»

Шаг 6«Найденный невыделенный нуль, обозначаем через $0'$ и выделяем содержащую его третью строку. Снимаем знак выделения с третьего столбца, в котором расположен 0^* из третьей строки» :

$$C_0 = \begin{matrix} \\ + \\ + \end{matrix} \begin{matrix} \quad + & & \\ & & \\ \end{matrix} \begin{pmatrix} 0^* & 5 & 7 & 0' \\ 2 & 0^* & 3 & 3 \\ 0' & 1 & 0^* & 1 \\ 3 & 0 & 5 & 3 \end{pmatrix}$$

Приступаем к шагу 4.

Шаг 4 «Среди невыделенных элементов матрицы C_0 нет нулей. Переходим

к шагу 9»

Шаг 9 «Среди невыделенных элементов матрицы Со находим минимальный элемент h = min {2, 3, 3, 3, 5, 3} = 2. Значение h = 2 вычитаем из элементов невыделенных строк с номерами 2 и 4 и прибавляем к элементам выделенного второго столбца. Вновь полученную эквивалентную матрицу стоимости обозначаем через C_0» :

$$C_0 = \begin{matrix} \\ + \\ \\ + \end{matrix} \begin{matrix} \begin{matrix} & & + & \end{matrix} \\ \begin{pmatrix} 0^* & 7 & 7 & 0' \\ 0' & 0^* & 1 & 1 \\ 0' & 3 & 0^* & 1 \\ 1 & 0 & 3 & 1 \end{pmatrix} \end{matrix}$$

Приступаем к шагу 5.

Шаг 5 «Во второй строке, содержащей невыделенный нуль ($c_{21}^0 = 0$), есть выделенный нуль ($c_{22}^0 = 0^*$). Переходим к шагу 6»

Шаг 6 «Найденный невыделенный нуль, обозначаем через $0'$ и выделяем содержащую его вторую строку. Снимаем знак выделения с второго столбца, в котором расположен 0^* из второй строки» :

$$C_0 = \begin{matrix} + \\ + \\ + \\ \\ \end{matrix} \begin{pmatrix} 0^* & 7 & 7 & 0' \\ 0' & 0^* & 1 & 1 \\ 0' & 3 & 0^* & 1 \\ 1 & 0 & 3 & 1 \end{pmatrix}$$

Приступаем к шагу 4.

Шаг 4 «Среди невыделенных элементов матрицы C_0 есть нуль ($c_{42}^0 = 0$) .Переходим к шагу 5»

Шаг 5 «Так как четвертая строка, содержащая невыделенный нуль, не содержит 0^* , то переходим к шагу 7»

Шаг 7 «Найденный невыделенный нуль обозначаем символом $0'$ и строим

L-цепочку (c_{42}^0, c_{22}^0, c_{21}^0, c_{11}^0, c_{14}^0)» :

$$C_0 = \begin{matrix} + \\ + \\ + \\ \end{matrix} \begin{pmatrix} 0^* & 7 & 7 & 0' \\ 0' & 0^* & 1 & 1 \\ 0' & 3 & 0^* & 1 \\ 1 & 0 & 3 & 1 \end{pmatrix}$$

Приступаем к шагу 8.

Шаг 8. В L-цепочке со всеми «символами, которые обозначены 0^* ,их всех заменяем нулями, а символы $0'$ заменяем на 0^* . Каждый $0'$, который находится вне L–цепочки заменяем нулем и убираем каждое выделение строк и столбцов. Найденной матрице стоимости даем обозначение через C_1»

$$C_0 = \begin{pmatrix} 0 & 7 & 7 & 0^* \\ 0^* & 0 & 1 & 1 \\ 0 & 3 & 0^* & 1 \\ 1 & 0^* & 3 & 1 \end{pmatrix}$$

Приступаем к шагу последующей итерации ($m = 2$).

Вторая итерация

Шаг 1 «Число элементов k в системе независимых нулей матрицы C_1равно четырем. А так как $k = 4 = n$, то переходим к шагу 2»

Шаг 2 «В соответствии с эквивалентной матрицей стоимости $C^* = C_1$ выписываем оптимальное решение рассматриваемой задачи, представленное в виде матрицы её переменных»

$$\tilde{X} = (x_{ij}^0) = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Так же значение целевой функции будет выглядеть как $F(x) = 19$.

Число допустимых назначений равно $n!$, где n число строк или столбцов матрицы стоимостей.

Стоимость M соответствует каждому недопустимому назначению.

Кроме того существуют задачи, для которых параметр c_{ij}, носит смысл эффективности выполнения i-ой работы j-м исполнителем, $i, j = \overline{1, n}$. В подобных задачах суммарная производительность реализации всех работ обязана быть наибольшей. Стало быть, если первоначальная задача считается задачей максимизации, то все элементы матрицы стоимости необходимо умножить на –1 и осуществить их сложение в достаточной мере большим числом с тем, чтобы матрица не включала в себя бы отрицательных элементов. Далее задачу необходимо решать, как задачу минимизации.

В случае если начальная матрица не считается квадратной, то есть потребность ввести нужное число строк и столбцов, а уже после их элементам приписать значения, которые определяются по условиям решаемой задачи.

3. DELPHI – СРЕДА РАЗРАБОТКИ ПРИКЛАДНЫХ ПРОГРАММ

Delphi представляется структурированным объектно–ориентированным языком программирования. Основной областью его применения считается создание прикладного программного обеспечения [6].

Среда Delphi существует как интегрированная оболочка разработчика, в ней включен набор специализированных программ, отвечающих за различные этапы разработки готового приложения

При помощи встроенного редактора исходных текстов получается первоначальный текст программы. Данный редактор специализирован и выделяется гибкими возможностями цветового выделения разных элементов текста программы, кроме того дает возможность оперативного ввода уже не раз встречавшихся конструкций. Левая панель редактора представляет из себя проводник, который дает возможность оперативного перемещения между частями исходного текста и по форме выстраиваемой программы. Основная характеристика программы заключается в удобном пользовательском интерфейсе, присутствие и простота применения необходимых частей управления.

В системе Delphi существует особый проектировщик форм, при поддержке которого, окна последующей программы представляются в виде форм. Проектировщик предоставляет возможность построения оптимальных размеров окон, расположения и выстраивания всех возможных элементов управления и меню, вставить готовые изображения, отметить подсказки, заголовки, подписи и так далее.

Delphi является достаточно многофункциональной и существенно сильной средой разработки приложений, RAD–оболочкой, в которой объединены последние подходы к программированию. Оболочка данного языка именуется революционной, так как написана на базе библиотеки VCL. В свою очередь Делфи поддерживает такого рода компоненты, как VBX. У Delphi визуальные компоненты показаны открытыми с целью надстройки и переписывания.

Кроме того данный язык содержит в себе локальный сервер Interbase для

разработки расширяемых на всех внешних SQL–серверах приложений. В среде Delphi для сохранения информации возможно применение такого формата файлов как .dbf и .db, те в свою очередь проектируют информационную систему для локальной машины. В таком случае когда, начнет применяться локальный InterBase for Windows 4.0, таким образом его приложение вне всякого сомнения будет использоваться и в составе более крупной системы с архитектурой клиент–сервер.

Delhi имеет следующие объектно–ориентированными свойства:

Инкапсуляция: «Класс представляет собой единство трех сущностей полей, методов и свойств. Объединение этих сущностей в единое целое и называется инкапсуляцией. Инкапсуляция позволяет во многом изолировать класс от остальных частей программы, сделать его «самодостаточным» для решения конкретной задачи. В результате класс всегда несет в себе некоторую функциональность. Например, класс TForm содержит (инкапсулирует в себе) все необходимое для создания окна Windows- программы, класс TMeto представляет собой полнофункциональный многострочный текстовый редактор, класс TTimer обеспечивает работу программы с таймером и т. д.

Инкапсуляция является мощным средством обмена готовыми к работе программными заготовками. Библиотека классов Delphi это, фактически, набор «кирпичиков», созданных программистами Borland для построения ваших программ» [19, с. 171].

«*Наследование* — это возможность создания иерархии классов, когда потомки наследуют все свойства своих предков, могут их изменять и добавлять новые. Свойства при наследовании повторно не описываются, что сокращает объем программы. Выделение общих черт различных классов в один класс-предок является мощным механизмом абстракции — ведь и любая наука начинается с абстрагирования и классификации, которые помогают справиться со сложностью рассматриваемой предметной области» [20, с. 176].

«Третьим китом, на котором стоит ООП, является *полиморфизм — возможность использовать и различных классах иерархии одно имя для обозначения сходным по смыслу действий и гибко выбирать требуемое действие во время выполнения программы*» [20, с. 176].

«От Visual Basic язык Delphi отличают строгая типизированность, позволяющая компилятору еще на этапе компиляции обнаружить многие ошибки, а также средства работы с указателями. Последнее дает возможность использовать так называемое раннее связывание с библиотеками типов в технологии COM. в то время как Visual Basic (и Java, в котором тоже не поддерживаются указатели) вынуждены при обращении к COM использовать более медленное позднее связывание и интерфейсы диспетчеризации.

Синтаксис C++ прямо-таки провоцирует создание запутанных программ, в которых трудно разобраться даже автору, в то время как простой и ясный синтаксис Delphi позволяет последнему претендовать на роль языка, идеально подходящего для описания алгоритма (недаром Паскаль происходит от использующегося для этих целей алгоритмического языка АЛГОЛ-60).

Если по каким-либо причинам возможности Delphi окажутся недостаточными, вы можете программировать на Ассемблере (машинно-зависимом языке программирования), который органично вплетен в Delphi.

Во всех случаях Delphi имеет самый быстрый среди продуктов подобного рода оптимизирующий компилятор, позволяющий создавать быстрые и относительно компактные программы» [19, с. 20].

3.1. Реализация венгерского метода решения задачи о назначениях на языке Delphi

При программировании любой программы необходимо иметь четкое представление, что такое алгоритм, как он работает и как его запрограммировать. От хорошо спроектированного алгоритма зависит сколько времени займет отладка приложения и поиск ошибок. Также алгоритм показывает структуру выполнения программы или части кода, что очень важно при модификации программы.

Наиболее распространенными способами представления алгоритмов являются словесные и блок-схемы. Словесный алгоритм венгерского метода описан в пункте 2 (ст. 8). Таким образом для максимально продуктивного и быстрого написания программы на базе словесного алгоритма я разработал блок-схему для данной задачи.

На которой четко наблюдаются все этапы выполнения венгерского алгоритма с целью решения задачи о назначениях [10]. Блок–схема показана на рисунках 3.1,3.2,3.3,3.4,3.5.

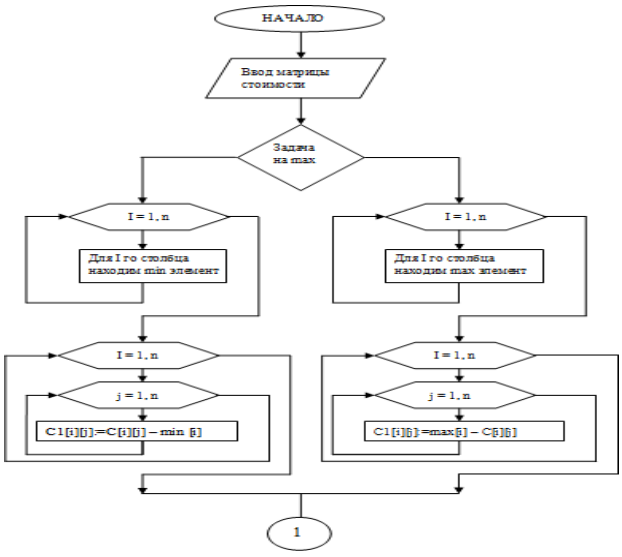

Рисунок 3.1 – Блок–схема венгерского алгоритма (подготовительный этап)

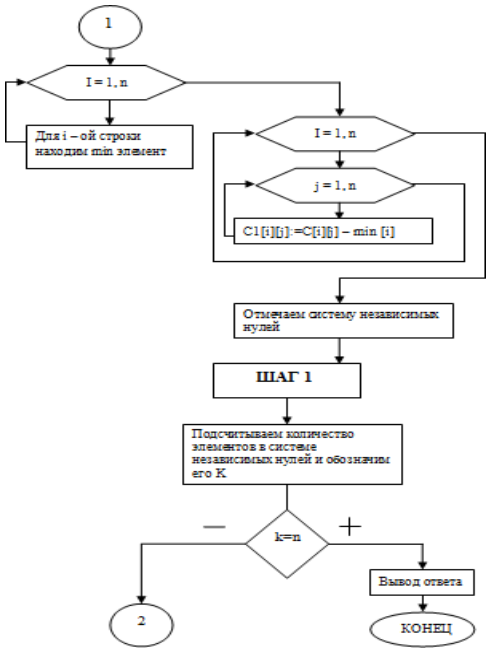

Рисунок 3.2 – Блок–схема венгерского алгоритма (шаг 1)

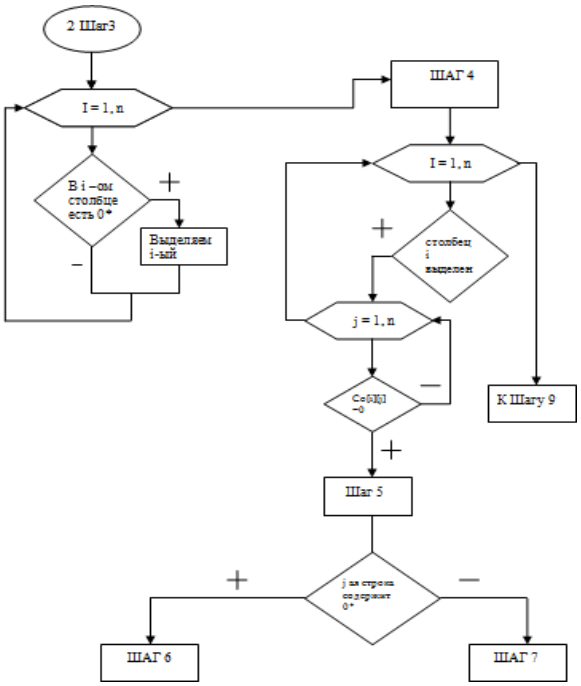

Рисунок 3.3 – Блок–схема венгерского алгоритма (шаг 2– шаг 7)

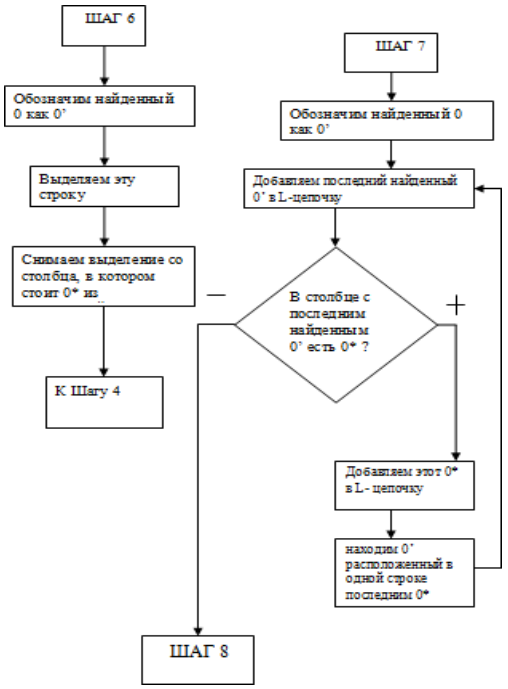

Рисунок 3.4 – Блок–схема венгерского алгоритма (шаг 6– шаг 8)

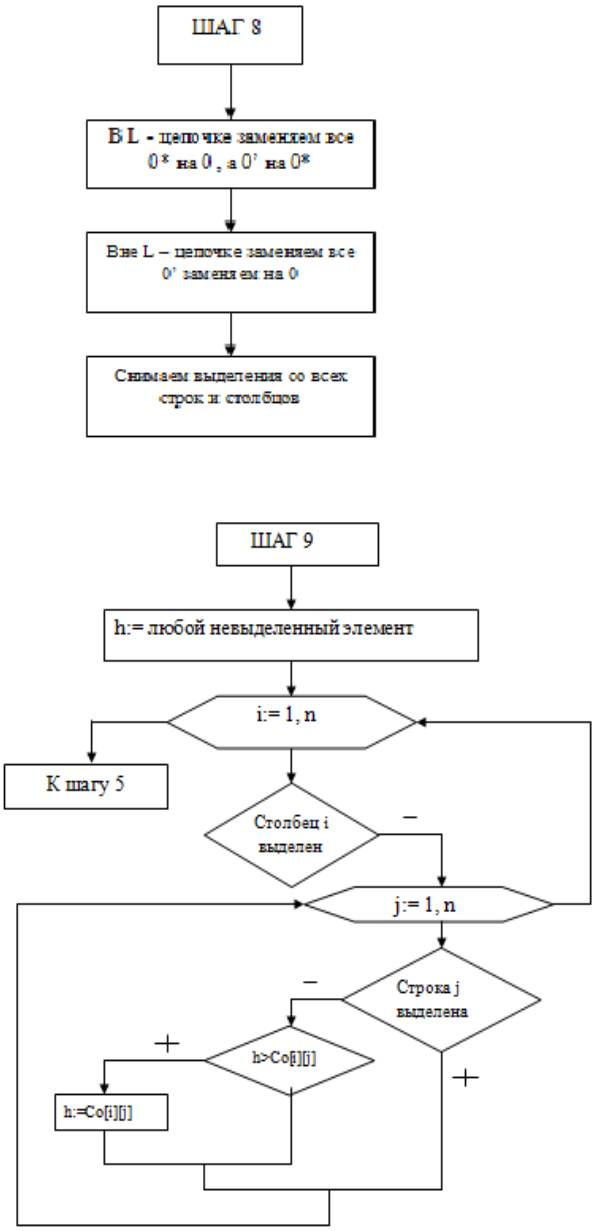

Рисунок 3.5 – Блок–схема венгерского алгоритма (шаг 8 – шаг 9)

«Венгерский метод решения задачи о назначениях» был выполнен с помощью «структурированного объектно–ориентированного языка программирования Delphi» Интерфейс программы изображен на рисунке 3.1.

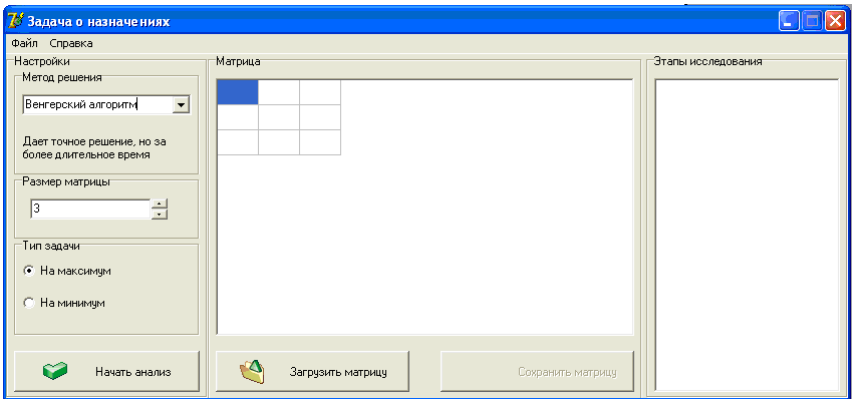

Рисунок 3.1 – Интерфейс программы

На основной панели расположена кнопка «Файл», в свою очередь может совершать такого рода операции, как: открыть; сохранить; начать анализ; выход.

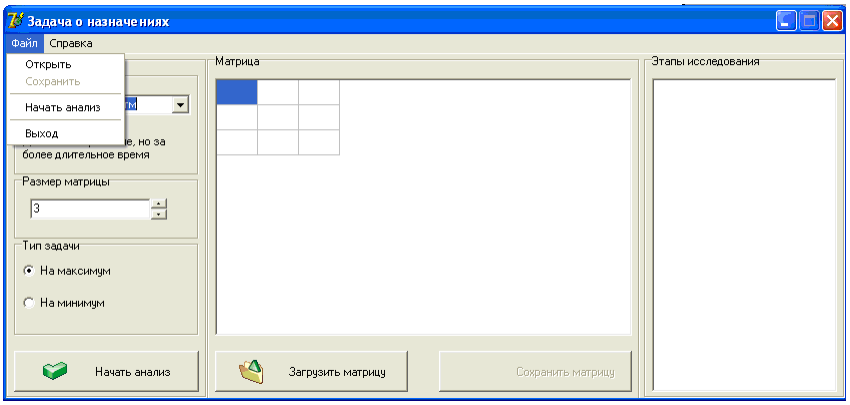

Рисунок 3.2 – Кнопка «Файл»

Рядом присутствует кнопка «Справка», которая позволяет узнать информацию о программе.

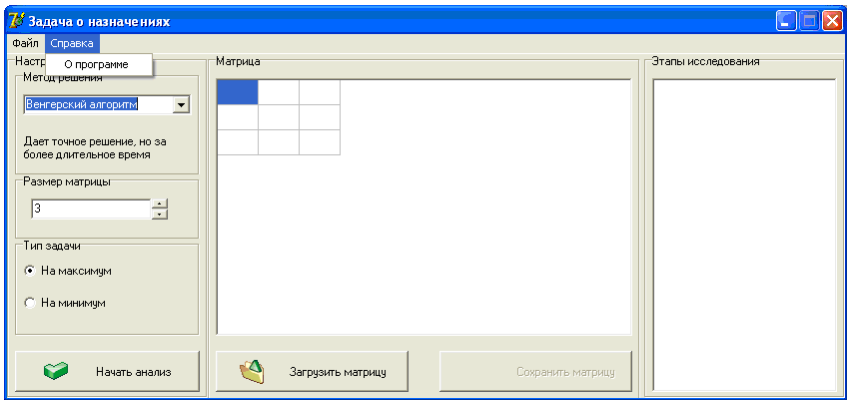

Рисунок 3.3 – Кнопка «Справка»

С целью нахождения решения задачи о назначениях венгерским методом определяем размерность матрицы и тип задачи. Для выполнения анализа нужно ввести матрицу размерность которая задана. Иначе при помощи нажатия на кнопку «Загрузить матрицу» загрузить её из файла.

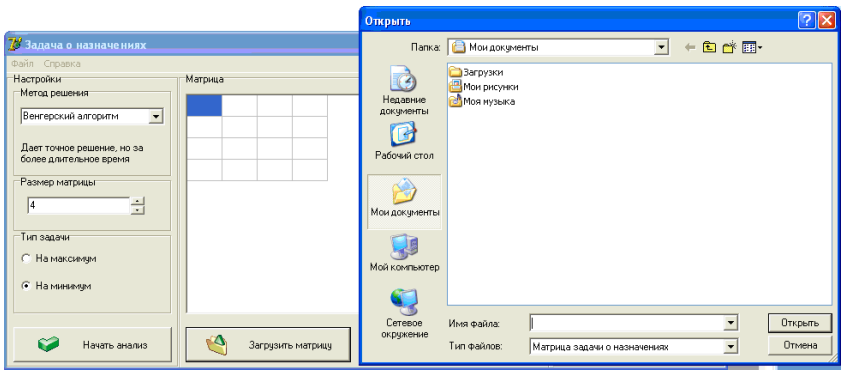

Рисунок 3.4 – Кнопка «Загрузить матрицу»

Таким образом существует функция, как «Сохранить матрицу», её возможности позволяют сохранить введенную матрицу в памяти компьютера, в виде текстового документа.

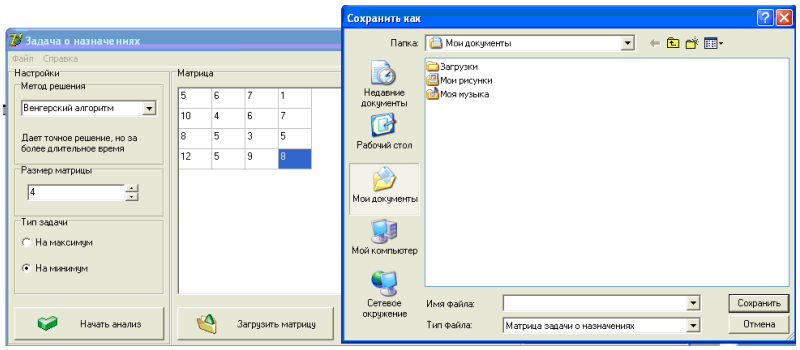

Рисунок 3.5 – Кнопка «Сохранить матрицу»

Также, после того как введена или загружена матрица следует приступить к анализу. Что бы это сделать нужно нажать на кнопку «Начать анализ».

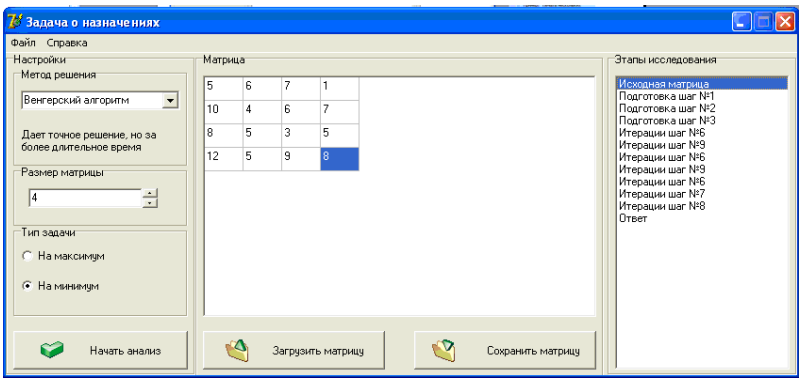

Рисунок 3.6 – Кнопка «Начать анализ»

После чего в правой части данной программы представляются этапы исследования, в которых показаны: исходная матрица, подготовительные шаги, итерационные шаги и заключительный ответ.

Покажем применение программы на реальном примере. Вводим

Исходную матрицу $C = \begin{matrix} 5 & 6 & 7 & 1 \\ 10 & 4 & 6 & 7 \\ 8 & 5 & 3 & 5 \\ 12 & 5 & 9 & 8 \end{matrix}$ (рисунок 3.6).

На подготовительном шаге №1 во всех столбцах матрицы найдем наименьший элемент и вычтем его из каждого элемента столбца. В итоге получаем равнозначную матрицу стоимости C', изображенную на рисунке 3.7.

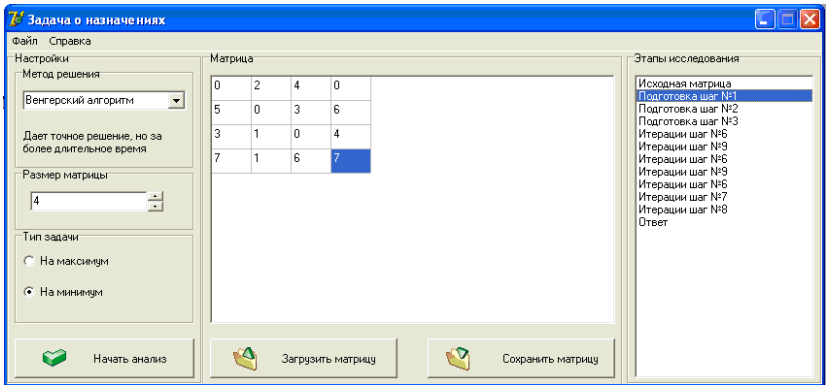

Рисунок 3.7 – Подготовительный шаг №1

В результате подготовительного шага №2 во всех строках матрицы находим наименьший элемент и отнимем его из каждого элемента строки, и таким образом строится равнозначная матрица стоимости C_0 , изображенную на рисунке 3.8.

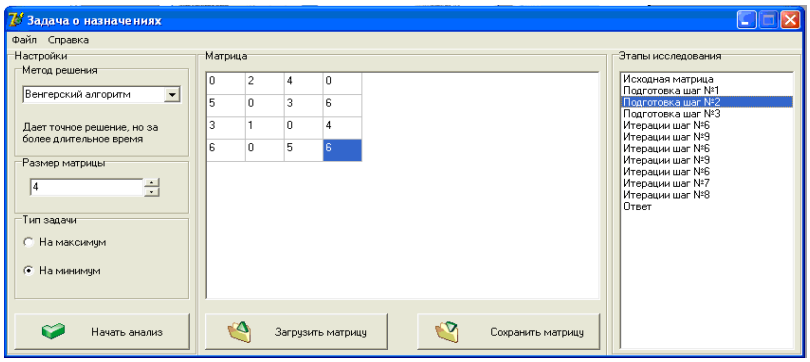

Рисунок 3.8 – Подготовительный шаг №2

Отмечаем систему независимых нулей на подготовительном шаге №3 (рисунок 3.9).

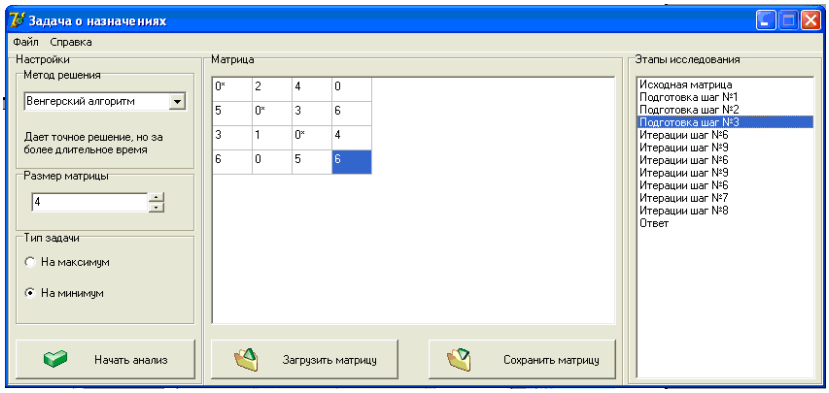

Рисунок 3.9 – Подготовительный шаг №3

Дальнейшим этапом венгерского метода будет итерация, складывающееся из 9 шагов.

Поскольку в матрице C_0 число независимых нулей ровняется $k = 3$, а $n = 4$ ищем еще один независимый нуль, при помощи №6 шаг итерации.

Шаг №6. «Найденный невыделенный нуль, обозначаем через $0'$ и выделяем содержащую его первую строку. Снимаем знак выделения с первого столбца, в котором расположен 0^* из первой строки» (рисунок 3.10).

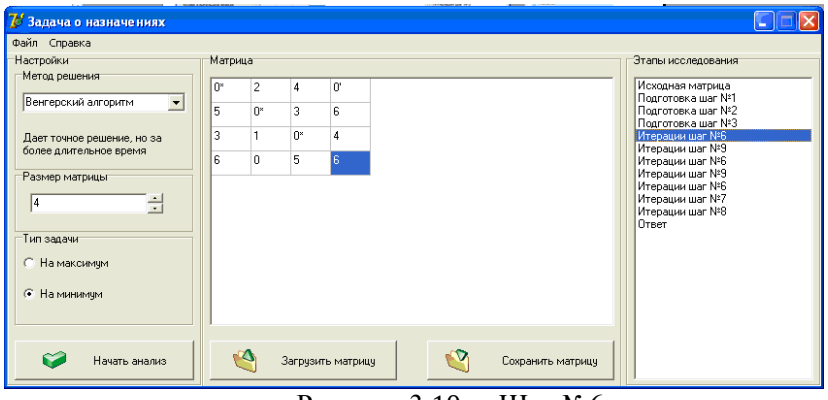

Рисунок 3.10 – Шаг №6

Шаг №9. «Среди невыделенных элементов матрицы Co находим минимальный элемент h = min {5, 3, 6, 6, 4, 6} = 3. Значение h = 3 вычитаем от

элементов невыделенных строк с номерами 2, 3, 4 и прибавляем к элементам выделенных столбцов с номерами 2, 3» (рисунок 3.11).

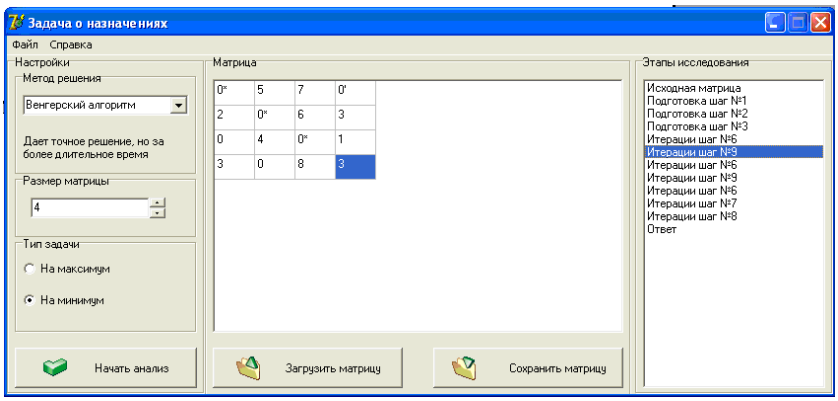

Рисунок 3.11 – Шаг №9

Шаг №6. «Найденный невыделенный нуль, обозначаем через 0′ и выделяем содержащую его третью строку. Снимаем знак выделения с третьего столбца, в котором расположен 0^* из третьей строки» (рисунок 3.12).

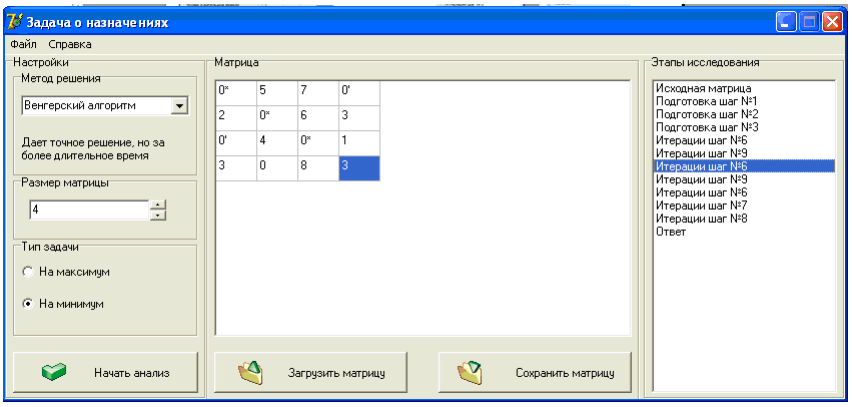

Рисунок 3.12 – Шаг №6

Шаг №9. «Среди невыделенных элементов матрицы Со находим минимальный элемент $h = \min\{2, 3, 3, 3, 5, 3\} = 2$. Значение $h = 2$ вычитаем из

элементов невыделенных строк с номерами 2 и 4 и прибавляем к элементам выделенного второго столбца» (рисунок 3.13).

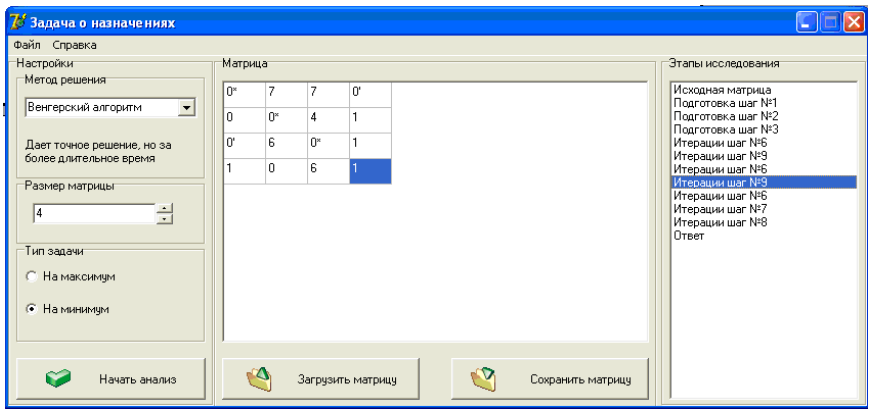

Рисунок 3.13 – Шаг №9

Шаг №6. «Найденный невыделенный нуль, обозначаем через $0'$ и выделяем содержащую его вторую строку. Снимаем знак выделения с второго столбца, в котором расположен 0^* из второй строки» (рисунок 3.13).

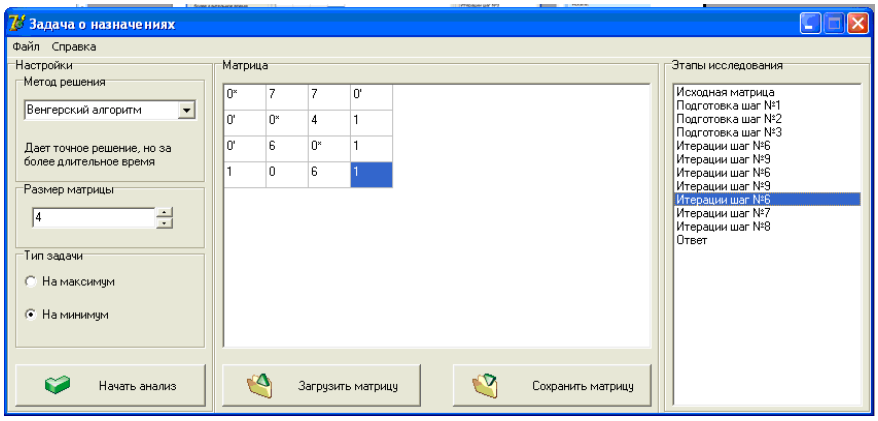

Рисунок 3.13 – Шаг №6

Шаг №7 «Найденный невыделенный нуль обозначаем символом $0'$ и строим L-цепочку (c_{42}^0, c_{22}^0, c_{21}^0, c_{11}^0, c_{14}^0)» (рисунок 3.14).

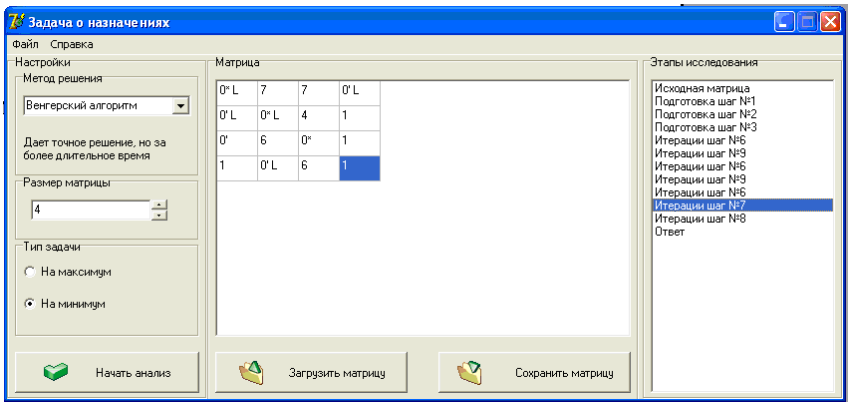

Рисунок 3.14 – Шаг №7

Шаг №8. В L–цепочке с каждыми «символами, которые обозначены 0^* ,их всех заменяем нулями, а символы $0'$ заменяем на 0^* . Каждый $0'$, который находится вне L–цепочки заменяем нулем и убираем каждое выделение строк и столбцов. Найденной матрице стоимости даем обозначение через C_1» (рисунок 3.14).

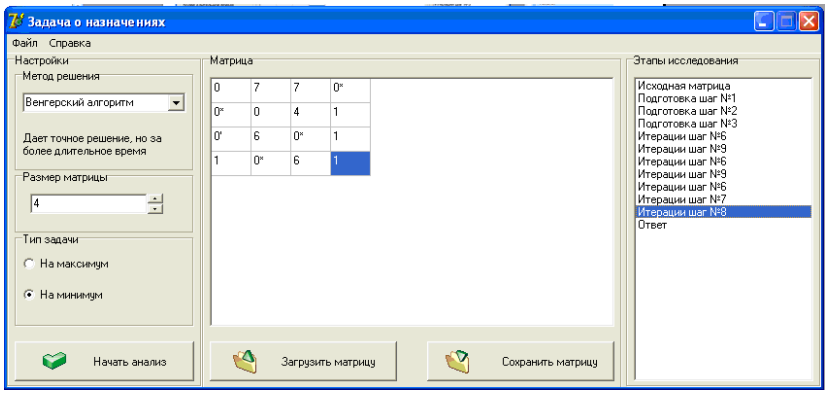

Рисунок 3.14 – Шаг №8

В итоге имеем оптимальное решение наблюдаемой задачи, показанной в виде матрицы. Таким образом значение целевой функции $F(x) = 19$ (рисунок 3.14).

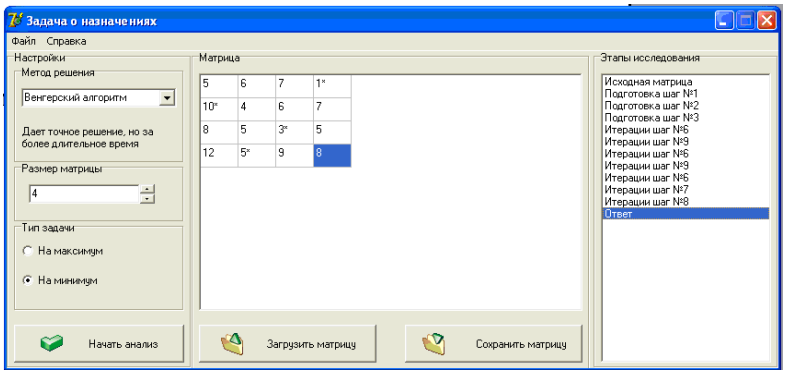

Рисунок 3.15 – Результат

В текущем разделе осуществлен венгерский метод решения задачи о назначениях на объектно–ориентированном языке Delphi.Работа алгоритма проверена на реальном примере, который показывает все этапы этого алгоритма. Венгерский метод существует как алгоритмом полиномиальной сложности, в связи с этим он находит решение любой задачи за допустимое время. А в свою очередь программа может быть применена с целью решения задачи о назначениях для любой матрицы стоимостей.

Заключение

К задачам дискретной оптимизации, а именно к задачам нахождения максимума или минимума той или иной функции на том или ином дискретном множестве, по формальным признакам сводятся большое количество оптимизационных задач, обладающих высоким народнохозяйственным значением. Такие задачи как организация производства, управления отраслью, проектирование техники и другие. К данным задачам относят и транспортную задачу.

Задача о назначениях, как особый случай обычной транспортной задачи обладает обширными практическими приложениями и не только к проблемам транспорта.

Применение симплекс-метода к задаче об назначениях малоэффективно, таким образом равно как каждое ее возможное базовое разрешение считается вырожденным. Характерные черты задачи о назначениях дали возможность создать продуктивный способ ее решения, знаменитый как венгерский метод. Данный алгоритм является алгоритмом оптимизации, он в свою очередь решает задачу о назначениях за полиномиальное время. Временная сложность представленного алгоритма не превышает $O(n^3)$.

В работе показана однокритериальная постановка задачи о назначениях, как в терминах теории матриц, так и в терминах теории графов.

Решение задачи осуществляется «венгерским алгоритмом. Формируется первоначальный план, не подходящий в общем случае под каждое условие задачи. Стало быть, происходит переход к новому плану, более близко подходящему к оптимальному. Постепенное использование такого приема за конечное количество итераций приводит к ответу решаемой задачи»

Преимуществом венгерского метода считается возможность давать оценку близости результата всех итераций к лучшему плану перевозок. Это дает возможность вести контроль процесс вычислений и завершить его при превышении определенных показателей. Это свойство значительно для задач у

которых большая размерность.

В работе показана компьютерная реализация венгерского метода решения задачи о назначениях в среде Delphi, которая является структурированным объектно–ориентированным языком программирования. Важной областью применения, которого представляется создание прикладного программного обеспечения.

Интерфейс программы показан на основной панели кнопками «Файл», «Справка». Программа дает возможность выбирать метод решения, размер матрицы и тип задачи (на максимум или на минимум), кроме того выполняет каждый этап венгерского метода с визуализацией промежуточных результатов, и конечного решения. Работа программы представлена на примере решения реальной задачи о назначениях. Код программы показан в приложении.

В современной среде развития предприятий, каждое предприятие пытается с минимальными расходами работать в сложившихся условиях с целью того ,чтобы получить высокий доход. Экономико–математические задачи о назначениях делают возможным нахождение оптимальных вариантов размещения одного кандидата на выполнение одной работы таким путем, что бы минимизировать расходы в сумме по выполнению совокупности работ группой исполнителей

Список используемой литературы и используемых источников

1. Волков И.К. Исследование операций / И.К. Волков, Е.А. Загоруйко.– М.: МГТУ им. Н.Э.Баумана, 2000.–436с.
2. Горлач Б.А. Исследование операций / Б.А. Горлач.– СПб.: Лань, 2013.–448с.
3. Гэри М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон .– М.: Мир, 1982.–416с.
4. Емеличев В.А. Лекции по теории графов / В.А. Емеличев, О.И. Мельников, В.И. Сарванов, Р.И. Тышкевич. – М.: Либроком, 2014.–384с.
5. Емеличев В.А. Сложность дискретных многокритериальных задач // Дискретная математика / В.А. Емеличев, В.А. Перепелица.– 1994.– Т.6, вып.1.– С.3 – 33.
6. Керман Н. Программирование и отладка в Dephi. Учебный курс / Н. Керман. – М.: Издательский дом «Вильямс», 2002.–672с.
7. Новиков Ф.А. Дискретная математика / Ф.А. Новиков. – СПб.: Питер, 2017.–496с.
8. Пестриков В.М. Delphi на примерах / В.М. Пестриков. – СПб.: БВХ – Петербург, 2005. – 496с.
9. Таха Хемди А. Введение в исследование операций / А. Таха Хемди .– М.: Издательский дом «Вильямс», 2005. – 912с.
10. Фаронов В.В. Delphi. Программирование на языке высокого уровня / В.В. Фаронов. – СПб.: Питер, 2013. –642с.
11. Наследов, А.Д. Математические методы. / А.Д. Наследов. – СПб: Речь, 2004. – 38 с.
12. Агальцов В.П. Математические методы в программировании. / В.П. Агальцов, И.В. Волдайская. – М.: ИД «ФОРУМ»: ИНФРА-М, 2006 г. – 224 с.
13. Цирель С.В. Венгерский способ. / С.В. Цирель. – Москва: УРСС,

2007. – 120 с.

14. Сакович В.А. Исследование операций (детерминированные методы и модели). / В.А. Сакович. – Мн.: Выш. шк., 1984. – 256 с.

15. Попов Ю.Д. Методические рекомендации для выполнения практических, лабораторных и самостоятельных работ по методам оптимизации и математического программирования на персональных компьютерах для студентов факультетов кибернетики, информационных технологий, компьютерных наук и менеджмента. / Ю.Д. Попов. – Киев: Издательско-полиграфический центр «Киевский університет», 2006. – 65 с.

16. Хомоненко А.Д. Самоучитель Delphi. 2-е изд.: БХВ-Петербург, 2008. – 576 с.

17. Россолов С.Ю. ПРИМЕНЕНИЕ МЕТОДА МАКА В РЕШЕНИИ ЗАДАЧ О НАЗНАЧЕНИЯХ. – СПб.: Журнал: Наука и образование сегодня, 2017. – 2 с.

18. Павловская Т.А. C/C++. Программирование на языке высокого уровня: [для вузов по направлению подгот. дипломир. специалистов "Информатика и вычисл. техника"]:Издательский дом "Питер", 2010 - 460 с.

19. Фаронов, В.В. Delphi. Программирование на языке высокого уровня / В.В. Фаронов. – Санкт-Петербург: Издательский дом "Питер", 2013 – 642 с.

20. Павловская, Т.А. C/C++. Программирование на языке высокого уровня: для вузов по направлению подгот. дипломир. специалистов "Информатика и вычисл. техника"/ Т.А. Павловская. – Санкт-Петербург: Издательский дом "Питер", 2010 – 460 с.

21. Марков, Е.П. Delphi 2005 для .NET/ Е.П. Марков. – Санкт-Петербург: БХВ-Петербург, 2005. – 896 с.

22. Атапин, В. Специальные главы математики. Множества, графы, комбинаторика / В. Атапин: Litres, 2018. – 82 с.

Приложение А

Фрагмент листинга программы, реализующий венгерский алгоритм решения задачи о назначениях на языке Delphi

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Grids, Buttons, ComCtrls, Menus;
type
TForm1 = class(TForm)
MainMenu1: TMainMenu;
N1: TMenuItem;
N2: TMenuItem;
GroupBox1: TGroupBox;
GroupBox2: TGroupBox;
GroupBox3: TGroupBox;
GroupBox4: TGroupBox;
ComboBox1: TComboBox;
Label1: TLabel;
GroupBox5: TGroupBox;
Edit1: TEdit;
UpDown1: TUpDown;
GroupBox6: TGroupBox;
RB1: TRadioButton;
RB2: TRadioButton;
BitBtn1: TBitBtn;
sg: TStringGrid;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
LB1: TListBox;
SaveDialog1: TSaveDialog;
procedure ComboBox1Change(Sender: TObject);
```

```
    procedure Edit1Change(Sender: TObject);
    procedure UpDown1Click(Sender: TObject; Button: TUDBtnType);

    n, i, j: integer;
    ForMax: boolean; // когда задача на максимум, то значение переменной true
    Original: array of array of integer; // для начала столбцы, после строки
    c, c1, c0:array of array of integer; // c1 = c&#39; в описании алгоритма
    MinElemsInCol, MinElemsInRow: array of integer;
    Cols, rows: array of boolean; // в случае если столбец или строка отмечены, то True в соответствующем элементе
    naznachen: array of integer; // указывает на какую работу назначен i–ый сотрудник массива
    nulls: array of integer; // демонстрирует в какой строке расположен помеченный звездой ноль в столбце, и какому номеру ячейки данного массива соответствует
    ShtrihNulls: array of TPoint; // демонстрирует в какой строке расположен помеченный штрихом ноль в столбце, и какому номеру ячейки данного массива соответствует
    SNCounter: integer; // счетчик элементов ShtrihNulls
    MarkedCol, MarkedRow: array of boolean;
    LChain: array of TPoint; // L–цепочка
    f: text;
    FOpen: boolean; // показывает открыт ли файл
    procedure Prohod;
    procedure Venger;
    procedure WriteAnswer;
    procedure podgotovka;
    procedure Iteration;
    procedure MakeLChain(q,w:integer);
    Procedure FindSNCounter;
    procedure CheckShtrih;
    procedure WriteToFile(L:boolean); // L – указывает на то, необходимо ли обозначать L–цепочку
    Function FindMaxInRow(q:integer):integer;
    Function FindMaxInCol(q:integer):integer;
```

```
Function FindMaxInColV(q:integer):integer; // Function
FindMaxInRow(q:integer):integer;

Function FindMinInRow(q:integer):integer; // Function FindMaxInCol(q:integer):integer
Function FindMinInCol(q:integer):integer;
Function FindMinInColV(q:integer):integer;
// Function FindMaxInCol(q:integer):integer;
Function FindMinInRowV(q:integer):integer;
function obespecheno: boolean;
Function IsInLChain(q,w:integer; Shtrih:boolean):boolean;
FUNCTION IsShtrih(q,w:integer):boolean;
implementation
{$R *.dfm}
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
case ComboBox1.ItemIndex of
0: Label1.Caption:='отдает ';
1: Label1.Caption:='отдает';
end;
end;
procedure TForm1.Edit1Change(Sender: TObject);
begin
if Edit1.Text<>'' then
begin
Sg.ColCount:=StrToInt(edit1.Text); //Sg.ColCount:=StrToInt(edit1.Text);
Sg.RowCount:=StrToInt(Edit1.Text); // Sg.RowCount:=StrToInt(Edit1.Text);
end;
end;
procedure TForm1.BitBtn1Click(Sender: TObject);
label ToEnd;
var cel: string;
begin
If FOpen then
begin
FOpen:=false;
```

```
CloseFile(f);
end;
LB1.Items.Clear;// инициализация массивов
n:=StrToInt(Edit1.Text);
SetLength(original, n);// SetLength(original, n);
SetLength(Cols, n); //SetLength(Cols, n);
SetLength(Rows, n);
SetLength(naznachen, n);
// SetLength(Cols, n);
// SetLength(Rows, n);
for i:=1 to n do
begin
SetLEngth(Original[i-1], n);
Cols[i-1]:=false;
Rows[i-1]:=false;
end;
if RB1.Checked then ForMax:=True
else ForMax:=False;
// получение первоначальных данных
for i:=1 to n do
For j:=1 to n do
begin
cel:=Sg.Cells[i-1, j-1];
if not(TryStrToInt(cel, original[i-1][j-1])) then
begin
ShowMessage('Введены неверные данные!!!');
Goto ToEnd;
end;
end;
case ComboBox1.ItemIndex of
0: Prohod;
1: Venger;
end;
WriteAnswer;
if Combobox1.ItemIndex=1 then
```

```
begin
procedure Prohod;
procedure Venger;
procedure WriteAnswer;
procedure podgotovka;
procedure Iteration;
procedure MakeLChain(q,w:integer);
Procedure FindSNCounter;
procedure CheckShtrih;
procedure WriteToFile(L:boolean); // L – указывает на то, необходимо ли обозначать
L–цепочку
Function FindMaxInRow(q:integer):integer;
Function FindMaxInCol(q:integer):integer;
Function FindMaxInColV(q:integer):integer; // Function
FindMaxInRow(q:integer):integer;
Function FindMinInRow(q:integer):integer; // Function FindMaxInCol(q:integer):integer
Function FindMinInCol(q:integer):integer;
Function FindMinInColV(q:integer):integer;
// Function FindMaxInCol(q:integer):integer;
Function FindMinInRowV(q:integer):integer;
function obespecheno: boolean;
Function IsInLChain(q,w:integer; Shtrih:boolean):boolean;
FUNCTION IsShtrih(q,w:integer):boolean;
implementation
{$R *.dfm}
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
case ComboBox1.ItemIndex of
0: Label1.Caption:='отдает ';
1: Label1.Caption:='отдает';
end;
end;
procedure TForm1.Edit1Change(Sender: TObject);
begin
if Edit1.Text<>'' then
```

```
begin
Sg.ColCount:=StrToInt(edit1.Text); //Sg.ColCount:=StrToInt(edit1.Text);
Sg.RowCount:=StrToInt(Edit1.Text); // Sg.RowCount:=StrToInt(Edit1.Text);
end;
procedure TForm1.BitBtn1Click(Sender: TObject);
label ToEnd;
var cel: string;
begin
If FOpen then
begin
FOpen:=false;
end;
for i:=1 to n do
for j:=1 to n do
writeln(f, Form1.Sg.cells[i-1, j-1]);
Form1.LB1.Items.Add('Ответ');
end;
Bitbtn3.Enabled:=true;
Form1.N3.Enabled:=true;
ToEnd:
end;
procedure Prohod;
Label ToStart, ToEnd;
var CurElemRow, CurElemCol:integer;
begin
FOpen:=false;
ToStart:
For i:=1 to n do
if not rows[i-1] then
begin
if ForMax then
Begin
CurElemRow:=FindMaxInRow(i-1); // демонстрирует в каком столбце ряда который выбран находится наибольший элемент
CurElemCol:=FindmaxInCol(CurElemRow);
```

```
if CurElemCol=i-1 {  }
then
begin
Rows[i-1]:=true;
Cols[CurElemRow]:=True;
naznachen[i-1]:=CurElemRow; // назначает работника на ту работу, где его наибольшая продуктивность
end;
End
Else
Begin
CurElemRow:=FindMinInRow(i-1); // демонстрирует в каком столбце ряда который выбран находится наибольший элемент
CurElemCol:=FindminInCol(CurElemRow);
if CurElemCol=i-1 {   }
then
begin
Rows[i-1]:=true;
Cols[CurElemRow]:=True;
naznachen[i-1]:=CurElemRow; // назначает работника на ту работу, где его наибольшая продуктивность
end; end; end;
if not obespecheno then goto ToStart;
end;
procedure Venger;
var i1, j1:integer;
FName:string;
begin
FName:=Paramstr(0);
i1:=Length(FName);
while FName[i1]<>'\' do
dec(i1);
FOpen:=true;
fname:=copy(FName, 1, i1)+'matrix.txt';
AssignFile(f, fname);
```

```
Rewrite(f);
writeln(f, n);
begin
Rows[i-1]:=true;
Cols[CurElemRow]:=True;
SetLength(MinElemsInCol, n);
SetLength(MinElemsInRow, n);
SetLength(c1[i1-1], n);
SetLength(c0[i1-1], n);
SetLength(c[i1-1], n);
SetLength(MarkedRow, n);
SetLength(MarkedCol, n);
SetLength(c1, n);
SetLength(c0, n);
SetLength(c, n);
for i1:=1 to n do
begin
SetLength(c1[i1-1], n);
SetLength(c0[i1-1], n);
SetLength(c[i1-1], n);
end;
For i1:=1 to n do
For j1:=1 to n do
begin
c[i1-1][j1-1]:=original[i1-1][j1-1];
writeln(f, c[i1-1][j1-1]);
if ForMax then
Begin
CurElemRow:=FindMaxInRow(i-1); // демонстрирует в каком столбце ряда который выбран находится наибольший элемент
CurElemCol:=FindmaxInCol(CurElemRow);
if CurElemCol=i-1 {  }
then
begin
Rows[i-1]:=true;
```

```
Cols[CurElemRow]:=True;
naznachen[i-1]:=CurElemRow; // назначает работника на ту работу, где его наибольшая продуктивность
end;
End
Else
Begin
CurElemRow:=FindMinInRow(i-1); // демонстрирует в каком столбце ряда который выбран находится наибольший элемент
CurElemCol:=FindminInCol(CurElemRow);
if CurElemCol=i-1 {   }
then
begin
Rows[i-1]:=true;
Cols[CurElemRow]:=True;
naznachen[i-1]:=CurElemRow; // назначает работника на ту работу, где его наибольшая продуктивность
end; end; end;
if not obespecheno then goto ToStart;
end;
procedure Venger;
var i1, j1:integer;
end;
Form1.LB1.Items.Add('Исходная матрица');
podgotovka;
iteration;
end;
Function FindMaxInRow(q:integer):integer; // возвращает номер столбца с наибольшим элементом в ряд который выбран
var i1, max:integer;
MaxRow: integer; // номер столбца с наибольшим элементом
Begin
max:=-1;
for i1:=1 to n do
if (original[i1-1][q]>max) and (not cols[i1-1]) then
```

```
begin
max:=original[i1-1][q];
MaxRow:=i1-1;
end;
result:=maxRow;
end;
Function FindMaxInCol(q:integer):integer; //
var i1, max:integer;
MaxCol:Integer;//SetLength(c1[i1-1], n); SetLength(c0[i1-1], n);SetLength(c[i1-1], n);
begin
max:=-1;
for i1:=1 to n do
if (original[q][i1-1]>max) and (not rows[i1-1]) then
begin
max:=original[q][i1-1];
MaxCol:=i1-1;
end;
Result:=maxCol;
end;
Function FindMaxInColV(q:integer):integer; //
var i1, max:integer;
MaxCol:Integer;
begin
max:=-1;
for i1:=1 to n do
if (original[q][i1-1]>max) then
begin
max:=original[q][i1-1];
MaxCol:=i1-1;
end;
Result:=maxCol;
end;
Function FindMinInRow(q:integer):integer;
var i1, min:integer;
MinRow: integer;
```

```
Begin
min:=60000;
for i1:=1 to n do
if (original[i1-1][q]<min) and (not cols[i1-1]) then
begin
min:=original[i1-1][q];
MinRow:=i1-1;
end;
result:=minRow;
end;
 Function FindMaxInColV(q:integer):integer; //
var i1, max:integer;
MaxCol:Integer;
begin
max:=-1;
for i1:=1 to n do
if (original[q][i1-1]>max) then
begin
max:=original[q][i1-1];
MaxCol:=i1-1;
end;
Function FindMinInRowV(q:integer):integer;
var i1, min:integer;
MinRow: integer;
begin
min:=c1[0][q];
MinRow:=0;
for i1:=1 to n do
if (c1[i1-1][q]<min) then
begin
min:=c1[i1-1][q];
MinRow:=i1-1;
end;
result:=minRow;
end;
```

```
Function FindMinInCol(q:integer):integer; //
var i1, min:integer;
begin
SetLength(c1[i1-1], n);
SetLength(c0[i1-1], n);
SetLength(c[i1-1], n);
end;
For i1:=1 to n do
For j1:=1 to n do
begin
c[i1-1][j1-1]:=original[i1-1][j1-1];
writeln(f, c[i1-1][j1-1]);
end;
Form1.LB1.Items.Add('Исходная матрица');
podgotovka;
iteration;
end;
Function FindMaxInRow(q:integer):integer; // возвращает номер столбца с
наибольшим элементом в ряд который выбран
var i1, max:integer;
var i1, max:integer;
MaxRow: integer; // номер столбца с наибольшим элементом
Begin
max:=-1;
for i1:=1 to n do
if (original[i1-1][q]>max) and (not cols[i1-1]) then
begin
max:=original[i1-1][q];
MaxRow:=i1-1;
end;
result:=maxRow;
end;
Function FindMaxInCol(q:integer):integer; //
var i1, max:integer;
MaxCol:Integer;//SetLength(c1[i1-1], n); SetLength(c0[i1-1], n);SetLength(c[i1-1], n);
```

```
begin
 max:=-1;
 for i1:=1 to n do
 if (original[q][i1-1]>max) and (not rows[i1-1]) then
 begin
 max:=original[q][i1-1];
 MaxCol:=i1-1;
 end;
 Result:=maxCol;
 end;
 Function FindMaxInColV(q:integer):integer; //
 var i1, max:integer;
 MaxCol:Integer;
 begin
 max:=-1;
 for i1:=1 to n do
 if (original[q][i1-1]>max) then
 begin
 max:=original[q][i1-1];
 MaxCol:=i1-1;
 end;
 Result:=maxCol;
 end;
 Function FindMinInRow(q:integer):integer;
 var i1, min:integer;
 MinRow: integer;
 Begin
 min:=60000;
 for i1:=1 to n do
 if (original[i1-1][q]<min) and (not cols[i1-1]) then
 begin
 min:=original[i1-1][q];
 MinRow:=i1-1;
 end;
 result:=minRow;
```

```
end;
 Function FindMaxInColV(q:integer):integer; //
var i1, max:integer;
MaxCol:Integer;
begin
max:=-1;
for i1:=1 to n do
if (original[q][i1-1]>max) then
begin
max:=original[q][i1-1];
MaxCol:=i1-1;
end;
MaxRow: integer; // номер столбца с наибольшим элементом
Begin
max:=-1;
for i1:=1 to n do
if (original[i1-1][q]>max) and (not cols[i1-1]) then
begin
max:=original[i1-1][q];
MaxRow:=i1-1;
end;
result:=maxRow;
end;
Function FindMaxInCol(q:integer):integer; //
var i1, max:integer;
MaxCol:Integer;//SetLength(c1[i1-1], n); SetLength(c0[i1-1], n);SetLength(c[i1-1], n);
begin
max:=-1;
for i1:=1 to n do
if (original[q][i1-1]>max) and (not rows[i1-1]) then
begin
MinCol:Integer;
begin
min:=60000;
for i1:=1 to n do
```

```
if (original[q][i1-1]<min) and (not rows[i1-1]) then
begin
min:=original[q][i1-1];
MinCol:=i1-1;
end;
Result:=minCol;
end;
Function FindMinInColV(q:integer):integer; //
var i1, min:integer;
MinCol:Integer;
begin
min:=original[q][0];
MinCol:=0;
for i1:=1 to n do
if (original[q][i1-1]<min) then
begin
min:=original[q][i1-1];
MinCol:=i1-1;
end;
Result:=minCol;
end; Function FindMinInCol(q:integer):integer; //
var i1, min:integer;
MinCol:Integer;
begin
min:=60000;
for i1:=1 to n do
if (original[q][i1-1]<min) and (not rows[i1-1]) then
begin
min:=original[q][i1-1];
MinCol:=i1-1;
end;
Result:=minCol;
end;
function obespecheno: boolean;
var i1:integer;
```

```
statusOk:boolean;
begin
StatusOk:=True;
for i1:=1 to n do
StatusOk:=StatusOk and rows[i1-1];
Result:=StatusOk;
end;
procedure WriteAnswer;
var i1, j1:integer;
CurString:string;
begin
//Form1.Sg.Font.Color:=clRed;
For i1:=1 to n do
begin
CurString:=Form1.Sg.Cells[naznachen[i1-1], i1-1]+'*';
Form1.Sg.Cells[naznachen[i1-1], i1-1]:=CurString;
end;
end;
procedure TForm1.BitBtn3Click(Sender: TObject);
end;
result:=maxRow;
end;
Function FindMaxInCol(q:integer):integer; //
var i1, max:integer;
MaxCol:Integer;//SetLength(c1[i1-1], n); SetLength(c0[i1-1], n);SetLength(c[i1-1], n);
begin
max:=-1;
for i1:=1 to n do
if (original[q][i1-1]>max) and (not rows[i1-1]) then
begin
MinCol:Integer;
begin
min:=60000;
for i1:=1 to n do
if (original[q][i1-1]<min) and (not rows[i1-1]) then
```

```
begin
min:=original[q][i1-1];
MinCol:=i1-1;
end;
Result:=minCol;
end;
Function FindMinInColV(q:integer):integer; //
var i1, min:integer;
MinCol:Integer;
begin
min:=original[q][0];
MinCol:=0;
for i1:=1 to n do
if (original[q][i1-1]<min) then
begin
min:=original[q][i1-1];
MinCol:=i1-1;
end;
Result:=minCol;
var FName:string;
FNaz:TextFile;
i1, j1:integer;
begin
if SaveDialog1.Execute then FName:=SaveDialog1.FileName+'.naz';
AssignFile(FNaz, FName);
Rewrite(Fnaz);
Writeln(fnaz, IntToStr(n));
for i1:=1 to n do // транспонированная матрица записывается в файл
begin
for j1:=1 to n do
Writeln(FNaz, IntToStr(original[i1-1][j1-1]));
//writeln(fnaz);
end;
CloseFile(Fnaz);
BitBtn3.Enabled:=false;
```

```
n3.Enabled:=false;
end;
procedure TForm1.BitBtn2Click(Sender: TObject);
var FName:string;
FNaz:TextFile;
i1, j1: integer;
CurNum, q:integer;
CurNumSt:string;
begin
if OpenDialog1.Execute then FName:=OpenDialog1.FileName;
AssignFile(Fnaz, FName);
Reset(FNaz);
Read(Fnaz, Curnum);
Edit1.Text:=IntToStr(CurNum);
//CurNum:=StrToINt(CurNumSt);
For i1:=1 to CurNum do
for j1:=1 to CurNum do
Begin
Read(FNaz, q);
Sg.Cells[i1-1, j1-1]:=IntToStr(q);
end;
CloseFile(FNaz);
end;
procedure podgotovka;
label back;
var i1, j1, k1:integer;
Ready:boolean;
begin
SetLength(nulls, n);
SetLength(ShtrihNulls, 0);
For i1:=1 to n do
begin
nulls[i1-1]:=-1;
//ShtrihNulls[i1-1].X:=-1;
//ShtrihNulls[i1-1].Y:=-1;
```

```
end;
// шаг1
// если на минимум
// ищим наименьшие элементы в столбцах
if Form1.RB1.Checked then
begin
for i1:=1 to n do
MinelemsInCol[i1-1]:=FindMaxInColV(i1-1);
// и создаем матрицу после 1 шага отнимаем наименьшие элементы
for i1:=1 to n do
for j1:=1 to n do
c1[i1-1][j1-1]:=c[i1-1][MinelemsInCol[i1-1]]-c[i1-1][j1-1];
end
else
begin
for i1:=1 to n do
MinelemsInCol[i1-1]:=FindMinInColV(i1-1);
// и создаем матрицу после 1 шага отнимаем наименьшие элементы
for i1:=1 to n do
for j1:=1 to n do
c1[i1-1][j1-1]:=c[i1-1][j1-1]-c[i1-1][MinelemsInCol[i1-1]];
end;
// делается запись матрицы c1 в файл по столбцам
for i1:=0 to (n-1) do
for j1:=0 to (n-1) do
writeln(f, c1[i1][j1]);
Form1.LB1.Items.Add('Подготовка шаг №1');
// шаг 2 - аналогичное для строк
For i1:=1 to n do
MinElemsInRow[i1-1]:=FindMinInRowV(i1-1);
For j1:=1 to n do
For i1:=1 to n do
c0[i1-1][j1-1]:=c1[i1-1][j1-1]-c1[MinElemsInRow[j1-1]][j1-1];
// запись в файл
writeToFile(false);
```

```
Form1.LB1.Items.Add('Подготовка шаг №2');
// шаг 3
for i1:=0 to (n-1) do
for j1:=0 to (n-1) do
writeln(f, c1[i1][j1]);
Form1.LB1.Items.Add
i1:=0;
j1:=0;
back:
if c0[i1][j1]=0 then
begin
Ready:=True;
For k1:=0 to (i1-1) do // делаем проверку на присутствие нулей раньше
If nulls[k1]=j1 then Ready:=False;
if Ready then
begin
nulls[i1]:=j1;
inc(i1);
j1:=-1;
end;
end;
if j1=(n-1) then
begin
j1:=-1;
inc(i1);
end;
inc(j1); // делаем проверку  следующего элемента в столбце
if i1<=(n-1) then goto Back;
// запись в файл
WriteToFile(False);
end;
if j1=(n-1) then
begin
j1:=-1;
inc(i1);
```

```
end;
inc(j1); // делаем проверку  следующего элемента в столбце
if i1<=(n-1) then goto Back;
// запись в файл
WriteToFile(False);
Form1.LB1.Items.Add('Подготовка шаг №3');
naznachen:=nil;
if FOpen then CloseFile(f);
Application.Terminate;
end;
procedure Iteration;
label ToEnd, Step1, Step2, Step3, Step4, Step5, Step6, Step7, Step8, Step9, Step4Back,
Step4Back1;
var NullCount, i1, j1, k1, k2:integer;
IsNull: boolean; // указывает, на наличие невыделенных нулей в столбцах на 4 шаге
minarray: array of integer;
minElem:integer;
MinArraySize:integer;
begin
Step1:
NullCount:=0;
for i1:=1 to n do
if Nulls[i1-1]<>-1 then inc(NullCount);
if NullCount=n then Goto Step2
else Goto Step3;
Step2: // преобразование завершено
for i1:=1 to n do
Naznachen[Nulls[i1-1]]:=i1-1;
Goto ToEnd;
Step3: // заполняем матрицу пометок столбцов, отталкиваясь от матрицы нулей
которые помеченны
For i1:=1 to n do
Begin
if Nulls[i1-1]<>-1 then inc(NullCount);
if NullCount=n then Goto Step2
```

```
else Goto Step3;
Step2: // преобразование завершено
for i1:=1 to n do
MarkedRow[i1-1]:=false;
MarkedCol[i1-1]:=false;
end;
For i1:=1 to n do
if Nulls[i1-1]<>-1 then MarkedCol[i1-1]:=true;
Step4: // поиск нулей
i1:=0;
IsNUll:=false;
Step4Back:
if not(MarkedCol[i1]) then
For j1:=0 to (n-1) do
if not (MarkedRow[j1]) then
if c0[i1][j1]=0 then
begin
k1:=j1; // k1 - строка, в которой содержится невыделенный нуль
k2:=i1; // k2 - столбец
GOTO Step5;
end;
inc(i1);
if i1=n then goto Step9 // если i1=n, это значит, что каждый столбец просмотрен
else goto Step4Back;
Step5: // шаг 5
// проверяем, наличие в строке с 0 элемент 0*
// тут нужно вновь искать нуль, который невыделенн
for i1:=0 to (n-1) do
if Nulls[i1]=k1 then Goto Step6;
// если нули найдены , то к шагу 7
Goto Step7; // заполняем матрицу пометок столбцов, отталкиваясь от матрицы нулей которые помеченны
Step6:
// на данный момент k1 включает в себя  номер столбца, а k2 - включает в себя номер строки с элементом 0'
```

```
// i1 включает в себя  номер столбца с элементом 0* из шага 5
//FindSNCounter;
SNCounter:=Length(ShtrihNulls)+1;
SetLength(ShtrihNulls, SNCounter);
ShtrihNulls[SNCounter-1].X:=k2;
ShtrihNulls[SNCounter-1].Y:=k1;
// делаем выделения нужных столбцов и строк
MarkedRow[k1]:=True;
MarkedCol[i1]:=False;
WriteToFile(False);
Form1.LB1.Items.Add('Итерации шаг №6');
Goto Step4;
Step7:
// для начала делаем обнуление цепочки
SetLength(ShtrihNulls, SNCounter);
ShtrihNulls[SNCounter-1].X:=k2;
ShtrihNulls[SNCounter-1].Y:=k1;
{SetLength(LCHain, n);
For i1:=0 to (n-1) do
begin
LChain[i1].X:=-1;
LChain[i1].Y:=-1;
end;} // полученный нуль который не выделен обозначим ';
SetLength(ShtrihNulls, Length(ShtrihNulls)+1);
ShtrihNulls[Length(ShtrihNulls)-1].X:=k2;
ShtrihNulls[Length(ShtrihNulls)-1].Y:=k1;
// теперь ищем цепочку
MakeLChain(k2, k1);
SetLength(LChain, Length(LChain)-1);
WriteToFile(True);
Form1.LB1.Items.Add('Итерации шаг №7');
SetLength(ShtrihNulls, Length(ShtrihNulls)+1);
ShtrihNulls[Length(ShtrihNulls)-1].X:=k2;
ShtrihNulls[Length(ShtrihNulls)-1].Y:=k1;
Step8: // замена 0* на 0'
```

```
for i1:=1 to Length(LChain) do
if odd(i1-1) then
Nulls[LChain[i1-1].x]:=-1; // это значит, что мы сняли с нуля символ '*'
for i1:= 1 to LEngth(LChain) do
if not(odd(i1-1)) then
Nulls[LChain[i1-1].x]:=LChain[i1-1].Y; // происходит замена символа 0' на 0* потому как в цепочке данный символ  на месте которое четно
// вне L-цепочки делаем замену всех 0' просто на 0
// необходимо просмотреть каждый 0' и проверить, находится ли он в цепочке
for i1:=1 to (Length(ShtrihNulls)) do
if ShtrihNulls[i1].X<>-1 then
if IsInLChain(ShtrihNulls[i1-1].X, ShtrihNulls[i1-1].Y, True) then
begin
ShtrihNulls[i1-1].X:=-1;
ShtrihNulls[i1-1].Y:=-1;
end;
// снимаем выделения строк
For i1:=0 to (n-1) do
Begin
MarkedCol[i1]:=false;
MarkedRow[i1]:=false;
end;// здесь необходимо каждый ненужный элемент из заштрихованных нулей убрать (которые-1)
CheckShtrih;
WriteToFile(False);
Form1.LB1.Items.Add('Итерации шаг №8');
Goto Step1;
Step9:
MinArraySize:=0;
For i1:=0 to (n-1) do
Begin
MarkedCol[i1]:=false;
MarkedRow[i1]:=false;
end;//
//FindSNCounter;
```

```
SNCounter:=Length(ShtrihNulls)+1;
SetLength(ShtrihNulls, SNCounter);
ShtrihNulls[SNCounter-1].X:=k2;
ShtrihNulls[SNCounter-1].Y:=k1;
// делаем выделения нужных столбцов и строк
MarkedRow[k1]:=True;
MarkedCol[i1]:=False;
WriteToFile(False);
Form1.LB1.Items.Add('Итерации шаг №6');
Goto Step4;
Step7:
// для начала делаем обнуление цепочки
SetLength(ShtrihNulls, SNCounter);
ShtrihNulls[SNCounter-1].X:=k2;
ShtrihNulls[SNCounter-1].Y:=k1;
{SetLength(LCHain, n);
For i1:=0 to (n-1) do
begin
LChain[i1].X:=-1;
LChain[i1].Y:=-1;
end;} // полученный нуль который не выделен обозначим ';
SetLength(ShtrihNulls, Length(ShtrihNulls)+1);
ShtrihNulls[Length(ShtrihNulls)-1].X:=k2;
ShtrihNulls[Length(ShtrihNulls)-1].Y:=k1;
For i1:=0 to (n-1) do // составим массив из невыделенных элементов
if not(markedCol[i1]) then
For j1:=0 to (n-1) do
if not(MarkedRow[j1]) then
begin
inc(MinArraySize);
SetLength(MinArray, MinArraySize);
MinArray[MinArraySize-1]:=c0[i1][j1];
end;
MinElem:=MinArray[0]; // и найдем в нем минимальный
For i1:=1 to (MinArraySize-1) do
```

```
if MinArray[i1]<MinElem then MinElem:=MinArray[i1]; // вычитаем минимальный элемент из всех невыделенных элементов
For i1:=0 to (n-1) do
if not(markedCol[i1]) then
For j1:=0 to (n-1) do
if not(MarkedRow[j1]) then
c0[i1][j1]:=c0[i1][j1]-MinElem; // прибавляем наименьший элемент к каждому выделенному ненулевому элементу
For i1:=0 to (n-1) do
if markedCol[i1] then
For j1:=0 to (n-1) do
if c0[i1][j1]<>0 then
c0[i1][j1]:=c0[i1][j1]+MinElem;
WriteToFile(False);
Form1.LB1.Items.Add('Итерации шаг №9');
// для корректности 5 шага нужно здесь находить 0
i1:=0;
IsNUll:=false;
Step4Back1:
if not(MarkedCol[i1]) then
For j1:=0 to (n-1) do
if not (MarkedRow[j1]) then
if c0[i1][j1]=0 then
begin
k1:=j1; // k1 - строка, содержащая невыделенный нуль
k2:=i1; // k2 - столбец
GOTO Step5;
end;
inc(i1);
if i1<>n then goto Step4Back1;
k1:=-2; // если произошло, что  0 не найден, то обозначим данный факт так
Goto Step5;
ToEnd:
end;
For i1:=0 to (n-1) do
```

```
if markedCol[i1] then
For j1:=0 to (n-1) do
if c0[i1][j1]<>0 then

procedure MakeLChain(q,w:integer); // q,w - номера столбца и строки (начиная с 0), в которых расположен 0'
Label OnceMore, ToEnd, L1;
var z, i2,j2,k2:integer;
Konec:boolean;
LCCounter:integer;
begin
SetLength(LChain, 1);
LChain[0].X:=q;
LChain[0].Y:=w;
i2:=1;
OnceMore:
inc(i2);
SetLength(LChain, i2);
MinArraySize:=0;
For i1:=0 to (n-1) do
Begin
MarkedCol[i1]:=false;
MarkedRow[i1]:=false;
end;//
For i1:=0 to (n-1) do // составим массив из невыделенных элементов
if not(markedCol[i1]) then
For j1:=0 to (n-1) do
if not(MarkedRow[j1]) then
begin
inc(MinArraySize);
SetLength(MinArray, MinArraySize);
MinArray[MinArraySize-1]:=c0[i1][j1];
end;
MinElem:=MinArray[0]; // и найдем в нем минимальный
For i1:=1 to (MinArraySize-1) do
```

```
if MinArray[i1]<MinElem then MinElem:=MinArray[i1]; // вычитаем минимальный элемент из всех невыделенных элементов
For i1:=0 to (n-1) do
if not(markedCol[i1]) then
For j1:=0 to (n-1) do
if not(MarkedRow[j1]) then
c0[i1][j1]:=c0[i1][j1]-MinElem; // прибавляем наименьший элемент к каждому выделенному ненулевому элементу
For i1:=0 to (n-1) do
if markedCol[i1] then
For j1:=0 to (n-1) do
if c0[i1][j1]<>0 then
c0[i1][j1]:=c0[i1][j1]+MinElem;
WriteToFile(False);
Form1.LB1.Items.Add('Итерации шаг №9');
// для корректности 5 шага нужно здесь находить 0
i1:=0;
IsNUll:=false;
If Nulls[LChain[i2-2].x]<>-1 then
Begin // записываем в цепочку координаты в матрице элемента 0*
LChain[i2-1].X:=LChain[i2-2].X;
LChain[i2-1].Y:=Nulls[LChain[i2-2].x]; // записываем в цепочку координаты в матрице элемента 0'
inc(i2);
SetLength(LChain, i2);
Lchain[i2-1].Y:=LChain[i2-2].Y;
For z:=0 to (Length(ShtrihNulls)-1) do
If ShtrihNulls[z].Y=Lchain[i2-2].Y then
Begin
SetLength(LChain, 1);
LChain[0].X:=q;
LChain[0].Y:=w;
i2:=1;
OnceMore:
inc(i2);
```

```
SetLength(LChain, i2);
Lchain[i2-1].X:=ShtrihNulls[z].x;
LChain[i2-1].Y:=ShtrihNulls[z].Y;
Goto OnceMore;
end;
end;
ToEnd:
end;
Function IsInLChain(q,w:integer; Shtrih:boolean):boolean; // есть ли элемент с координатами q,w в цепочке
var i3, j3:integer;
Res:boolean;
begin
res:=false;
If Shtrih then
Begin
OnceMore:
inc(i2);
SetLength(LChain, i2);
MinArraySize:=0;
For i1:=0 to (n-1) do
Begin
MarkedCol[i1]:=false;
MarkedRow[i1]:=false;
end;//
For i1:=0 to (n-1) do // составим массив из невыделенных элементов
if not(markedCol[i1]) then
For j1:=0 to (n-1) do
if not(MarkedRow[j1]) then
begin
inc(MinArraySize);
SetLength(MinArray, MinArraySize);
MinArray[MinArraySize-1]:=c0[i1][j1];
end;
MinElem:=MinArray[0]; // и найдем в нем минимальный
```

```
For i1:=1 to (MinArraySize-1) do
if MinArray[i1]<MinElem then MinElem:=MinArray[i1]; // вычитаем минимальный элемент из всех невыделенных элементов
For i1:=0 to (n-1) do
if not(markedCol[i1]) then
For j1:=0 to (n-1) do
if not(MarkedRow[j1]) then
c0[i1][j1]:=c0[i1][j1]-MinElem; // прибавляем наименьший элемент к каждому выделенному ненулевому элементу
For i1:=0 to (n-1) do
if markedCol[i1] then
For j1:=0 to (n-1) do
if c0[i1][j1]<>0 then
c0[i1][j1]:=c0[i1][j1]+MinElem;
WriteToFile(False);
Form1.LB1.Items.Add('Итерации шаг №9');
For i3:=0 to (Length(LChain)-1) do
if (i3 mod 2 = 0) and (LChain[i3].X=q) and (LChain[i3].Y=w) then Res:=True
end
else
begin
For i3:=0 to (Length(LChain)-1) do
if (i3 mod 2 = 1) and (LChain[i3].X=q) and (LChain[i3].Y=w) then Res:=True
end;
Result:=res;
end;
Procedure FindSNCounter;
var i2:integer;
begin
for i2:=0 to n*n do
if ShtrihNulls[i2].X=-1 then
begin
SNCounter:=i2;
exit;
end;
```

```
end;
procedure CheckShtrih;
label Back, Back2;
var i2, j2:integer;
begin
i2:=0;
Back:
inc(i2);
Back2:
if length(ShtrihNulls)>0 then
if ShtrihNulls[i2-1].X<0 then
begin
for j2:=(i2) to (Length(ShtrihNulls)-1) do
begin
ShtrihNulls[j2-1].X:=ShtrihNulls[j2].X;
ShtrihNulls[j2-1].y:=ShtrihNulls[j2].y;
end;
SetLength(ShtrihNulls, Length(ShtrihNulls)-1);
Goto Back2; // если осуществлена замена не нужного элемента, таким образом на его месте возможно окажется тоже ненужный элемент, в связи с этим делаем проверку не увеличивая счетчик
end;
If i2<Length(ShtrihNulls) then Goto Back;
end;
procedure WriteToFile(L:boolean); // L - указывает на то, нужно ли отмечать L-цепочку
var i4, j4:integer;
begin
if L then
begin
for i4:=1 to n do
for j4:=1 to n do
if Nulls[i4-1]=j4-1 then
begin
if IsInLChain(i4-1, j4-1, False) then writeln(f, c0[i4-1][j4-1], '*', ' L')
```

```
else
Writeln(f, c0[i4-1][j4-1], '*');
end
else
if IsShtrih(i4-1, j4-1) then
begin
if IsInLChain(i4-1, j4-1, True) then writeln(f, c0[i4-1][j4-1], #39, ' L')
else
Writeln(f, c0[i4-1][j4-1], #39);
end
else
Writeln(f, c0[i4-1][j4-1]);
end
else
for i4:=1 to n do
for j4:=1 to n do
if Nulls[i4-1]=j4-1 then Writeln(f, c0[i4-1][j4-1], '*')
else
if IsShtrih(i4-1, j4-1) then Writeln(f, c0[i4-1][j4-1], #39)
else
Writeln(f, c0[i4-1][j4-1]);
end;
FUNCTION IsShtrih(q,w:integer):boolean; // если элемент в столбце q и строке w 0' -
то возвращает истину
var res:boolean;
i5:integer;
begin
res:=false;
for i5:=0 to (Length(ShtrihNulls)-1) do
if (ShtrihNulls[i5].X=q) and (ShtrihNulls[i5].Y=w) then res:=true;
Result:=res;
end;
procedure TForm1.LB1Click(Sender: TObject);
var i6, j6:integer;
Matrix: array of string;
```

```
Element: string;
begin
//ShowMessage(IntToStr(LB1.ItemIndex));
SetLength(Matrix, n*n);
CloseFile(f);
Reset(f);
Readln(f, Element);
for i6:=1 to Form1.LB1.ItemIndex do // нам необходимо считать элементы до
нужных, в связи с этим начинаем с 1, а не с 0
for j6:=1 to (n*n) do
Readln(f, Element);
For i6:=0 to (n*n-1) do
Begin
procedure TForm1.LB1Click(Sender: TObject);
var i6, j6:integer;
Matrix: array of string;
Element: string;
begin
Readln(f, Element);
Matrix[i6]:=Element;
Sg.Cells[i6 div n, i6 mod n]:=Element;
end;
end;
procedure TForm1.N9Click(Sender: TObject);
begin
ShowMessage('о программе');
end;
procedure TForm1.N8Click(Sender: TObject);
begin
application.Terminate;
end;
procedure TForm1.N6Click(Sender: TObject);
begin
BitBtn1.Click;
end;
```

```
procedure TForm1.N3Click(Sender: TObject);
begin
BitBtn2.Click;
end;
procedure TForm1.N4Click(Sender: TObject);
begin
BitBtn3.Click;
end;
procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
if Edit1.Text = '' then Edit1.Text := '0';
if Button = btNext then Edit1.Text := IntToStr(StrToInt(Edit1.Text)+1)
else Edit1.Text := IntToStr(StrToInt(Edit1.Text)-1);
end;
end.
```