УДК 004:62-52:004.03

Н.О. Комлевая, канд. техн. наук

# АНАЛИЗ МЕТОДОВ РАЗРАБОТКИ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ОБУЧЕНИЯ АЛГОРИТМАМ НА ГРАФАХ

*Розглядаються питання розробки автоматизованої системи навчання алгоритмам знаходження оптимального шляху на графові. Проведено аналіз і порівняння існуючих методів розробки програмних систем. Виконано проектування навчальної системи з використанням схем алгоритмів і за допомогою алгоритмічної алгебри як інструмента для розробки.*

*Рассматриваются вопросы разработки автоматизированной системы обучения алгоритмам нахождения оптимального пути на графе. Проведен анализ и сравнение существующих методов разработки программных систем. Выполнено проектирование обучающей системы с использованием схем алгоритмов и при помощи алгоритмической алгебры как инструмента для разработки.*

*The problem to develop automated system of teaching algorithms to find out the optimal way on graphs are considered. Analysis and comparison of current methods to work out program systems have been conducted. Projecting of teaching system with the help of usage of algorithmic schemas and also with a help of algorithmic algebra as a tool to work out has been done.*

**Введение.** При современной наполняемости учебных групп педагог физически лишен возможности осуществлять принцип индивидуального подхода к учащимся для управления их познавательной деятельностью. Использование специальных обучающих программ существенно снижает поток информации от индивидуального педагога к учащемуся, значительно облегчает управляемость и поднимает эффективность обучения. Благодаря существенному увеличению потока информации от обучающей программы к учащемуся значительно повышается уровень обучения, сокращается время и разброс в успеваемости различных учащихся.

Переход на программное управление процессом обучения требует разработки новых методов исследования педагогического процесса для получения не столько качественных его характеристик, сколько объективных количественных мер и оценок, которые поддаются счету, статистической обработке и могут составить надежную основу для нормирования и аналитического описания [1].

В рамках разработки подобных методов предлагается рассмотрение вопросов автоматизации процесса обучения работе с алгоритмами на графах. Для этого необходимо провести анализ и выбор средств разработки и выполнить с их использованием проектирование автоматизированной системы обучения.

**Анализ и сравнение методов разработки автоматизированных систем.** Как известно, для решения любой задачи можно использовать различные методы, каждый из которых обладает рядом особенностей и возможностей. Первым этапом решения задачи идет формулирование проблемы, в результате чего следует постановка задачи. На основании анализа поставленной задачи выбирается соответствующий математический аппарат.

Далее можно выбрать один из следующих путей решения: использование схем алгоритма (СА); словесное описание алгоритма; алгоритмические алгебры (АА). Выбор одного из первых двух путей позволяет лишь упростить и формализовать дальнейшее написание программы. Выбор АА в качестве инструмента для разработки автоматизированной системы предоставляет широкий круг возможностей, а именно: автоматический синтез программы; построение экспертной системы символьной обработки; формальный синтез программного кода; описание целого множества алгоритмов.

Схемы алгоритмов – конечный связный ориентированный граф $G = \{A, V\}$, вершины которого $a_i \in A, i = \overline{1, N}$ соответствуют опе-

раторам, а дуги $v_k = (a_i, a_j) \in V, k = \overline{1, M}, i, j = \overline{1, N}$ задают порядок следования вершин (операторов) алгоритма, где $N=|A|$ – число вершин графа, $M=|V|$ – число дуг. В более широком смысле, вершинам графа соответствуют не только операторные вершины, но и условные, начальная и конечная вершины и т.д.

Схемы алгоритмов ориентированы на детализацию алгоритмов с целью их реализации в виде компьютерных программ, специализированных устройств, производственно-технологических процессов или других форм трудовой и общественной деятельности.

Наряду с аппаратом СА особый интерес представляет описание алгоритмов посредством соответствующих математических формул. К формульным представлениям применимы весьма глубокие аналитические преобразования, нацеленные на улучшение качества алгоритмов и их оптимизацию по выбранным критериям.

Для формализации постановки задач целесообразно использовать логико-функциональные модели [4]. Под моделью понимается система $\tilde{M}$ = < $A$; СИГНп >, где $A$ – основа (множество данных), СИГНп $\{\pi_i \mid i \in I\}$ – сигнатура, состоящая из предикатов $\pi_i$, определенных на множестве $A$.

Пусть $A$ – произвольное множество и $f(x_1, x_2, ..., x_n)$ – функция, определенная на множестве $A$ и принимающая значения в этом же множестве так, что $f(a_1, a_2, ..., a_n) = a$, где $a_i \in A$; $i = 1, 2, ..., n$ и $a \in A$. Функцию $f(x_1, ..., x_n)$ называют операцией, определенной на множестве $A$. Пусть на множестве $A$ задана совокупность операций $\Omega = F_i (x_1, x_2, ..., x_n)$, $i = 1, 2, ..., к$. Универсальной алгеброй является система $\tilde{A}$ = < $A$; $\Omega$ > где множество $A$ — основа алгебры, $\Omega$ – сигнатура данной алгебры.

Использование данного подхода в дальнейшем позволит выполнять преобразования аналитических представлений алгоритмов для повышения их функционирования во времени, использовать специализированные программные средства для синтеза программ. При этом проектирование осуществляется на основе взаимосвязи алгоритмов, структур данных и памяти. Для этого определяются используемые структуры данных, операции и предикаты для решения задачи.

В качестве элементарных блоков в процессе проектирования будем использовать следующие схемы, адекватные основным конструкциям структурного программирования: композиция $A * B$ – последовательное применение операторов $A$ и $B$; альтернатива $([u]A, B)$ – если условие $u$ – истинно, то выполнить оператор $A$, иначе – $B$; цикл $\{[u] A\}$ – пока не станет истинным условие $u$, выполнять оператор $A$; где $A$, $B$ – операторные переменные, $u$ – логическая переменная. Как обычно, составные логические условия будем получать посредством известных булевых операций «и» (конъюнкция $u \wedge u'$), «или» (дизъюнкция $u \vee u'$) и «не» (отрицание $\bar{u}$). Далее, в процессе конструирования алгоритмов, будут использоваться переменные операторы $A, B, C,...$ и переменные условия $u$ с индексами, или без них.

Охарактеризуем некоторые проблемы конструирования автоматизированных систем с использованием метода АА [3].

Первая проблема – способ представления знаний об алгоритмах. В состав таких знаний входит классификация алгоритмов, условия их эффективной применимости, собственно запись текста алгоритмов, сведения о возможных преобразованиях алгоритмов при их привязке к условиям задачи и многое другое.

Вторая проблема – организация сборки алгоритмов. Она также решается с помощью системного тезауруса, возможности параметризации идентификаторов понятий и привязки к ним дополнительных атрибутов. Если пользователь не ограничивается алгоритмами, рекомендуемыми системой, то ему предоставляются возможности конструировать свои схемы. При этом из тезауруса выбирается или вводится главное понятие, характеризующее алгоритм в целом, и, возможно, набор конкретизирующих его равенств. Это составляет заготовку будущей схемы.

Затем инициируется процесс связывания равенств, в ходе которого для каждого из включенных в схему понятий ищется конкретизирующее равенство: сначала в ней самой, а затем в базе знаний. В случае неоднозначности оценивается эффективность вариантов

продолжений. При этом пользователь может отказаться от найденных равенств, выбрать из них подходящие или заменить на новые равенства. Понятия, оставшиеся без конкретизации, являются элементарными.

Такая организация сборки делает полезной разработку и накопление в базе набора абстрактных схем, из которых путем соответствующей интерпретации понятий и связывания равенств синтезируются СА для выбранной предметной области.

Следующая проблема – отладка разрабатываемых алгоритмов и автоматизация процесса синтеза программ. Для синтеза программы (на целевом языке) по составленной схеме достаточно отобразить ее управляющие конструкции: «*», «ЕСЛИ», «ЦИКЛ» в соответствующие операторы целевого языка и подставить вместо элементарных понятий их реализации на том же языке. При этом основная задача состоит в сохранении связи между структурой синтезированной программы и исходной СА. Такая связь необходима для трассировки и отладки в терминах понятий из тезауруса. Один из путей решения этой задачи состоит в том, чтобы отлаживать алгоритмы на уровне схем системы алгоритмических алгебр. Это возможно, так как в состав тезауруса могут включаться не только операторы и условия, но и объекты. В простейшем случае объекты отображаются в структуру данных и используются в качестве параметров для операторов и условий.

**Разработка системы обучения алгоритмам на графах.** Рассмотрим в качестве примера алгоритмы нахождения оптимального пути на взвешенном графе. Под оптимальным будем понимать кратчайший и длиннейший пути.

При работе с графами часто приходится выполнять некоторое действие по одному разу с каждой из вершин графа. Подобный обход можно совершать двумя различными способами. При обходе в глубину проход по выбранному пути осуществляется настолько глубоко, насколько это возможно, а при обходе в ширину выполняется равномерное продвижение вдоль всех возможных направлений. Рассмотрим более подробно алгоритмы обхода в ширину, называемые волновыми, в которых от рассматриваемой вершины исходит виртуальная волна, захватывающая смежные с ней вершины. Исходный граф может быть произвольным графом, задаваемым весами дуг-переходов.

В качестве исходных данных обучающая программа использует следующую информацию об исследуемом графе:

размер графа (число его вершин-состояний);

информацию о дугах-переходах: пара вершин, соединяемых дугой; направление перехода между вершинами; вес дуги-перехода;

информацию о начальной и конечной вершинах, между которыми определяется оптимальный путь.

При этом:

ребро, соединяющее две вершины, заменяется парой двух взаимно направленных дуг с теми же весами, что и вес исходного ребра;

если отсутствует информация о весе дуги, он считается равным единице.

В текущем сеансе работы обучающая программа использует информацию о точке входа в программу. Перечень возможных точек входа:

начало алгоритма;

пересчет весов вершин, начиная с первой вершины графа;

пересчет весов вершин, начиная с произвольной вершины графа, кроме первой (задается номер вершины в соответствии с нумерацией вершин при задании графа);

проверка достижимости конечной вершины;

выделение дуг-претендентов, начиная с первой вершины графа;

выделение дуг-претендентов, начиная с произвольной вершины графа, кроме первой;

сборка оптимального пути из выделенных дуг-претендентов, вычисление длины этого пути.

Программа реализует следующие функции:

ввод перечисленных выше данных о графе и процессе обучения. Предоставление обучающемуся информации о ранее рассмотренных графах;

запрос пользователя на точку входа в обучающую программу. В случае, когда точ-

ка входа не является начальной, система выполняет все предыдущие шаги автоматически;

запрос пользователя на режим работы: демонстрационный или рабочий;

в случае выбора пользователем демонстрационного режима работы программа с высокой степенью детализации (в текстовом и графическом виде) отображает работу алгоритма. Осуществляется демонстрация работы алгоритма на текущем шаге с последующим запросом: продолжить, повторить текущий шаг или остановиться;

в случае выбора пользователем рабочего режима работы программа предоставляет ему возможность выполнить текущий шаг. Если промежуточные результаты, полученные на текущем шаге, являются правильными, осуществляется запрос: перейти к следующему шагу или остановиться. Если же полученные результаты неверны, то переход к следующему шагу невозможен: пользователь может попытаться повторить текущий шаг, остановиться или же перейти к демонстрационному режиму на текущем шаге.

Данные обучающей системы отображаются на экране монитора и сохраняются в базе данных в следующем виде: номер сеанса; идентификатор пользователя; дату работы; размер исходного графа; перечень этапов, успешно пройденных в рабочем режиме.

Структурная схема работы системы обучения приведена на рисунке.

Здесь двойные линии показывают взаимодействие внешних лиц с системой, одинарные стрелки – передачу управления от одного блока к другому и информационные связи.

Использование данной информации позволит формировать выборки и анализировать работу с обучающей программой при наличии всех успешно пройденных этапов работы алгоритма по следующим критериям:

идентификатор пользователя;

размер исходного графа.

Обучающая система поддерживает работу с двумя категориями лиц – пользователь (обучающийся) и эксперт. Рассмотрим более подробно работу системы при обучении пользователя.

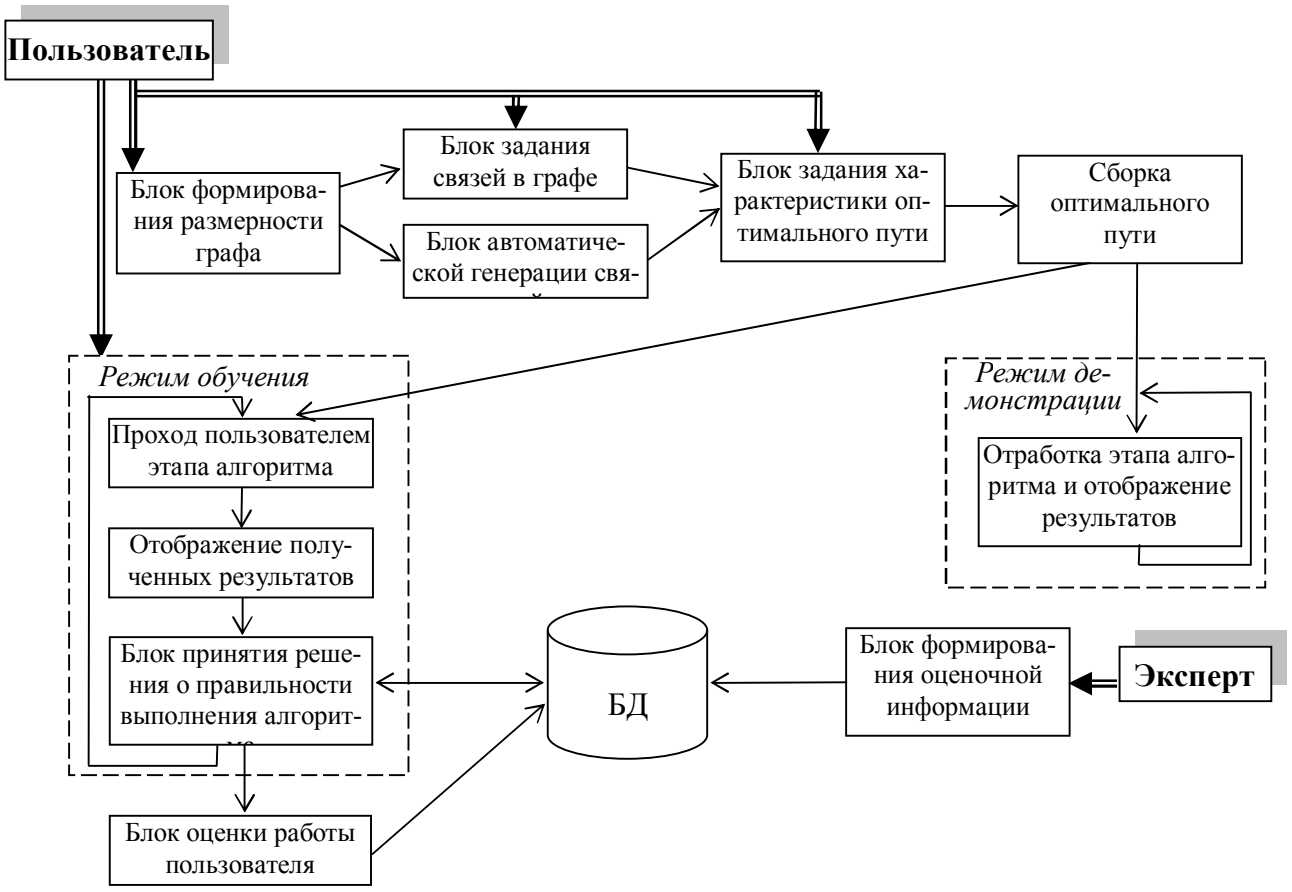


Рисунок. Структурная схема системы обучения работе с алгоритмами поиска оптимального пути

Начальным этапом функционирования системы обучения является выполнение блоков формирования размерности графа, задания связей в графе и определения характеристики оптимального пути – кратчайший или длиннейший путь.

Далее обучающая программа, согласно правилам работы волнового алгоритма для взвешенного графа, рассчитывает оптимальный путь с сохранением всей промежуточной информации: веса вершин при пересчете, выбранные дуги-претенденты, включенные в путь дуги и т.д.

Алгоритм поиска кратчайшего или длиннейшего пути выполняется в три прохода [2]. На первом вычисляются и пересчитываются веса всех вершин, включая конечную. На втором проходе вычисляются «претенденты» на оптимальный путь, т.е. дуги, которые могут составить этот путь (или несколько путей с одинаковым весом). На третьем проходе из дуг-претендентов составляются конкретные оптимальные пути. Задача составления пути из дуг-претендентов является переборной. Здесь важно учесть все возможные варианты перебора.

По окончании работы алгоритма выполняется блок оценки работы пользователя, результаты сохраняются в базе данных. Значения параметров для оценки знаний пользователя определяются экспертом.

Оценка работы осуществляется по формуле

$$O=\sum_{i=1}^{n}(k\,Rez-pm),$$

где $n$ – число этапов алгоритма; $k$ – коэффициент сложности этапа; $Rez$ – результат прохождения этапа (0 – отрицательный, 1 – положительный); $p$ – количество ошибочных итераций; $m$ – величина штрафа ($m$>0, если $Rez$=1, $m$=0, если $Rez$=0).

Рассмотрим построение оптимального пути с использованием СА. Для этого выделим следующие структуры данных – основные и вспомогательные:

$n$ – размерность (число вершин) графа;

$A[n \times n]$ – матрица смежности, задающая взвешенный ориентированный граф;

$V[n]$ – одномерный массив весов вершин графа;

$i$ – индекс (номер) текущей рассматриваемой вершины графа;

$M[n]$ – одномерный массив номеров вершин, изменивших вес;

$k$ – индекс последнего элемента в массиве $M$; $k$=0, если $M$ пуст;

$D[n \times n]$ – матрица с весами дуг-претендентов на оптимальный путь;

*Length* – длина (в процессе работы текущая, в конце – итоговая) оптимального пути;

*Row*$[n]$ – одномерный массив номеров вершин, через которые проходит оптимальный путь, $n$ – максимальный размер массива, реально размер может быть меньше.

В процессе работы алгоритма можно выделить следующие подпрограммы:

процедура *PREPARE* – начальная подготовка;

процедура *SELECT* – выбор из массива $M$ текущей вершины с последующей корректировкой содержимого массива $M$;

процедура *RECOUNT* – пересчет весов вершин для исходов текущей вершины графа;

процедура *ROWS* – формирование дуг-претендентов на включение в оптимальный путь;

процедура *WAYS* – сборка оптимального пути;

функция *NULLS* – проверка наличия нерассмотренных дуг-претендентов на включение их в оптимальный путь.

В связи с тем, что графическое представление СА для данного алгоритма является громоздким, приведем его словесное описание.

Согласно СА, вначале вызывается процедура *PREPARE*, которая выполняет начальную подготовку.

Далее проверяется размер массива $M$. Если элементы в нем отсутствуют, то происходит переход к выполнению процедуры *ROWS*. Иначе вызывается процедура *SELECT*, выбирающая из массива $M$ текущую вершину. Затем процедура *RECOUNT* пересчитывает веса вершин для исходов текущей вершины графа и переходит опять на проверку наличия элементов в массиве номеров вершин $M$. Далее процедура *ROWS* выделяет дуги-претенденты на включение в оптимальный путь. Наконец, процедура

*WAYS* формирует из дуг-претендентов оптимальный путь (один или несколько с одинаковыми показателями).

Рассмотрим теперь более подробно работу используемых алгоритмом подпрограмм.

В подпрограмме *PREPARE* пользователем вначале вводится размерность графа $n$ и определяются связи в графе и их веса с помощью матрицы смежности $A[n \times n]$. Затем первой вершине (начальной) присваивается вес, равный 0, а всем остальным нерассмотренным вершинам – веса, равные $\infty$. Первая вершина вносится в массив $M$, она считается текущей рассматриваемой вершиной.

В подпрограмме *SELECT* вначале из массива $M$ выбирается первый элемент $M[1]$. Его значение присваивается переменной $i$, которая хранит номер текущей рассматриваемой вершины графа.

Далее в цикле со второго по $k$-й элемент (последний элемент в массиве $M$) все элементы массива сдвигаются на один – выполняется операция $M[p\text{-}1]=M[p]$. Этим достигается корректировка содержимого массива $M$, так как если элемент уже взят как текущий, его номер должен быть исключен из массива. При этом число элементов $k$ сокращается на единицу.

Подпрограмма *RECOUNT* организует цикл по всем вершинам графа $p = 1{:}n$. Для каждой вершины определяется наличие дуги, ведущей из текущей вершины с индексом $i$ в данную вершину: $A[i, p] \neq 0$. Если такая дуга есть, то вес ее отличен от 0 и в матрице смежности будет ненулевое значение. Если же связи нет, то в матрице смежности проставлен 0. При отсутствии связи между указанными вершинами организуется переход к следующей по номеру вершине. Если же связь есть, то вычисляется новый возможный вес вершины как сумма веса $i$-й вершины и дуги, ведущей из $i$-й вершины в рассматриваемую. Полученная сумма сохраняется в переменной *temp*.

Далее сравниваются значения переменной *temp* и текущего веса рассматриваемой вершины $V[p]$. Если $temp < V[p]$, то новый вес вершины будет равен $temp$: $V[p] = temp$. При этом размер массива $M$ увеличивается на единицу и в него включается номер вершины $p$, изменившей свой вес.

Подпрограмма *ROWS* выделяет дуги, соединяющие вершины такие, что вес дуги равен разности весов соответствующих вершин. Для этого организуется цикл по вершинам и определяются разности между их весами, данные разности сохраняются в переменной *temp*. Значение переменной *temp* сравнивается со значением в соответствующей ячейке матрицы смежности $A$. В случае совпадения значение из матрицы смежности $A$ копируется в соответствующую ячейку матрицы дуг-претендентов $D$. Если же веса не равны, то в ячейке матрицы $D$ будет записан 0.

Подпрограмма *NULLS* вызывается в процедуре *WAYS*. Вначале некоторой переменной $f$ присваивается значение, равное 0. Затем делается полный перебор по матрице $D$, содержащей дуги-претенденты. Если нерассмотренных дуг в матрице не осталось, то функция возвращает результат равный 0. Иначе переменная $f$ принимает значение равное 1, и это значение возвращается в качестве результата работы функции.

Согласно СА подпрограммы *WAYS*, вначале с помощью функции *NULLS* проверяется наличие в матрице $D$ нерассмотренных дуг. Если все дуги-претенденты рассмотрены и произведены попытки задействовать их для формирования оптимального пути, то работа подпрограммы завершается. Иначе делаются начальные установки для построения оптимального пути: переменной *Length*, которая служит для хранения длины пути, присваивается значение 0, переменной $ip$ – начальной точке отсчета для построения пути – присваивается 1, в массив *Row* заносится 1 (вершина номер 1 задействована в пути): $j = 1$, $Row[j] = 1$. Переменной *count*, предназначенной для подсчета числа пройденных вершин, присваивается значение 1.

Далее проверяется наличие дуг, ведущих из текущей вершины с номером $ip$, к другой вершине. При наличии такой вершины ее номер помещается в следующую позицию массива *Row*, длина дуги ($ip$, $jp$) добавляется к значению переменной *Length*. Рассмотренная дуга в массиве $D$ обнуляется и счетчик числа пройденных вершин *count* увеличивается на 1. Анализируется значение номера вершины, в который входит текущая дуга.

Если номер равен *n* (конечная вершина), то достигнут конец пути, выводится значение содержимого массива *Row* и длина пути *Length*.

Иначе проверяется значение переменной *count*. Если оно равно *n*, то выполнено критическое число шагов алгоритма и по правилам построения оптимального пути формирование текущего пути заканчивается. Такие правила построения для кратчайшего пути очевидны, а для длиннейшего алгоритмом рассматриваются только элементарные или простые пути. При наличии нерассмотренных дуг производится следующая попытка. Иначе (при *count* ≠ *n*) продолжается построение оптимального пути.

Таким образом, разработанные структуры данных и СА позволили формализовать и упростить процесс создания автоматизированной системы обучения.

Представим теперь задачу поиска оптимального пути с использованием выражений АА.

Представление задачи поиска:

*SOLUTION* := {*PREPARE* * {[*k*=0] *SELECT* * *RECOUNT*} * *ROWS* * *WAYS*}
*PREPARE* := {*A* * *p*:=2 * {[*p*>*n*] *V*[*p*]=*GM* * *p*++} * *V*[1]=0 * *k*:=1 * *i*:=1 * *M*[*u*]=*i*}
*SELECT* := {*i*:=*M*[1] * *p*:=2 * {[*p*>*k*] *M*[*p*-1]=*M*[*p*] * *p*++} * *k*--}
*RECOUNT* := {*p*:=1 * {[*p*>*n*] * (*A*[*i*,*p*]<>0] *temp*:=*V*[*i*]+*A*[*i*,*p*] * ([*temp*<*V*[*p*]] *V*[*p*]:=*temp* * *k*++ * *M*[*k*]:=*p E*)}
*ROWS* := *j*=1 * {[*j*=*n*] *j*2=1 * {[*j*2=*n*] *temp*:=*V*[*j*2]-*V*[*j*] * ([A[j,j2]=temp] *D*[*j*,*j*2] = *A*[*j*,*j*2], *D*[*j*,*j*2]=0) * *j*2++} * *j*++}
*NULLS* := {*f*:=0 * *j*1:=1 * {[*j*1>*n*] * *j*2:=1 * {[*j*2>*n*] ([*D*[*j*1,*j*2]>0] *f*:=1 * *j*1:=*n*+1 * *j*2:=*n*+2) *j*1++ * *j*2++)} * возврат *f*}
*WAYS* := {*res*:=*NULLS**([*res*=0] *Length* := 0**ip*:=1**j*:=1**Row*[*j*]:=1* *count*:=0* *jp*=1 *{[*jp*=*n*] ([*D*[*ip*,*jp*]>0] *j*=*j*+1 * *Row*[*j*]=*jp* * *Length* = *Length* + *D*[*ip*,*jp*] * *D*[*ip*,*jp*]:=0 * *count*++ * ([*count*=*n*]*jp*:=*n*,*E*),*E*)} * *OUT*(*Row*[1..*j*]) * *OUT*(*Length*),*E*) }

**Выводы.** В результате проведенного исследования выполнены анализ и сравнение существующих методов разработки программных систем. Спроектирована обучающая система для работы с алгоритмами поиска оптимального пути. Использованы два подхода: метод разработки с помощью СА и метод с использованием АА.

В дальнейшем планируется использование выражений АА для автоматического синтеза программы поиска оптимального пути.

Список использованной литературы

1. Беспалько В.П. Программированное обучение. Дидактические основы /Беспалько В.П. – М.: Высш. шк., 1970. – 300 с.

2. Паулин О.Н. Основы теории алгоритмов / Паулин О.Н. Уч. пособие. – Одесса: Автограф, 2002. – 188 с.

3. Пригожев А.С. Автоматизированный перевод исходных текстов программ / Пригожев А.С. // Електромаш.та ел.обладнання. – 2009. – № 72. – С. 222 – 225.

4. Цейтлин Г.Е.. Введение в алгоритмику / Цейтлин Г.Е. – К.: Наука, 2002. – 216 с.

Получено 04.10.2010

Комлевая
Наталия Олеговна,
канд. техн. наук, доцент
каф. «Системного
программного
обеспечения» Одесск. нац.
политехн. ун-та,
т.: (048)734-85-66