УДК 004.8: 004.421.2

Д. А. ГАЙДАЧУК, А. Г. ЧУХРАЙ

Национальный аэрокосмический университет им. Н. Е. Жуковского «ХАИ», Украина

МЕТОД СТРУКТУРНОГО ДИАГНОСТИРОВАНИЯ СТУДЕНЧЕСКИХ КОМПЬЮТЕРНЫХ ПРОГРАММ

Данная работа посвящена исследованию вопроса обучения студентов при помощи интеллектуальных компьютерных обучающих программ (ИКОП) в сфере алгоритмизации и программирования. Рассматривается один из вопросов, возникающих при обучении профессиональным умениям алгоритмизации и программирования, а именно – как сопоставлять эталонную программу, хранящуюся в ИКОП, с программой, составленной обучаемым. Предлагается метод структурного диагностирования компьютерных программ, разработанных студентами, который основан на модификации существующего алгоритма сравнения двух структур данных, а именно m-арных деревьев. Такими деревьями могут быть абстрактные синтаксические деревья.

Ключевые слова: *интеллектуальная компьютерная обучающая программа, m-арное дерево, расстояние редактирования, след, программа, разработанная студентом, эталонная программа.*

Введение

На сегодняшний день одной из актуальных научно-прикладных проблем является проблема обеспечения эффективного индивидуального обучения профессиональным умениям. Так, из-за ограниченности психофизиологических способностей человека один педагог не может адаптивно обучать каждого обучаемого в группе из двадцати-тридцати человек.

Перспективным выходом из сложившейся ситуации может стать разработка и внедрение интеллектуальных компьютерных обучающих программ (ИКОП). Такие программы могут обладать практически неограниченными ресурсами и характеризоваться высоким быстродействием.

В данной работе рассматривается один из вопросов, возникающих при обучении профессиональным умениям алгоритмизации и программирования, а именно – как сопоставлять эталонную программу, хранящуюся в ИКОП с программой, составленной обучаемым [1, 2].

Подразумевается, что такие обучающие программы сначала проверяют решение, написанное пользователем, на некотором языке программирования на множестве тестов. А именно, сравниваются при различных входных данных выходные данные алгоритма, написанного пользователем, с выходными данными эталонного алгоритма. Если на каком-то тесте произойдет ошибка, а обучаемый не сможет ее самостоятельно исправить, обучающая программа должна будет определить, в каком месте она возникла, и помочь ее устранить.

Некоторые ошибки в коде обучаемого могут быть выявлены при помощи статического анализа [3]. В настоящее время существуют разработки, подходящие для получения навыков в области программирования [4], но также необходимы средства, которые бы позволяли приобретать и улучшать навыки обучаемым в области алгоритмизации.

Среди существующих программ, обучающие программы или обучают программированию за счет изучения особенностей синтаксиса языка, но не алгоритмизации [5], или определяют корректность программы только при помощи выходных данных для заданных входных тестов [5, 6], или для обучения используют не эталонные решения, а результаты компиляции и запуска программы [7].

Один из способов решения этих проблем – использование структурного диагностирования. Он может быть применен как для анализа кода обучаемого на предмет наличия или отсутствия ключевых слов, так и для нахождения «проблемного» участка кода, на основе эталонного решения задания, предоставленного обучаемому. Структурный анализ может осуществляться на уровне представления кода с помощью абстрактных синтаксических деревьев [8].

Целью данной работы является реализация алгоритма, позволяющего сопоставлять две древовидные структуры данных и находить минимальную последовательность операций редактирования для преобразования одного дерева в другое. Данный алгоритм является одним из методов структурного диагностирования компьютерных программ.

1. Постановка задачи

Требуется создать метод нахождения минимального расстояния редактирования и следа между двумя m-арными деревьями. Данный метод отлича-

ется от распараллеленного варианта алгоритма, опубликованного в 1989 году Шашей (Dennis Shasha) и Жангом (Kaizhong Zhang), который находит только расстояние между деревьями [9]. Также разрабатываемый метод должен превосходить алгоритм Шаши-Жанга по объему используемых ресурсов и времени выполнения.

Данный метод должен иметь возможность в дальнейшем быть использованным для диагностирования кода обучаемого на структурном уровне при помощи сравнения абстрактных синтаксических деревьев эталонного кода и кода обучаемого.

2. Описание метода

При описании алгоритма будут использованы следующие термины.

Дерево (m-арное) – это связный ациклический граф (количество исходящих или входящих в вершину ребер не превышает (m+1)).

Расстояние редактирования – это взвешенное число операций редактирования, которые осуществляют преобразование одного дерева в другое.

След – это последовательность операций редактирования, обеспечивающих преобразование одного дерева в другое.

Лист (мн. – «листья») – это узел дерева, не имеющий потомков.

Корень – это узел дерева, не имеющий предков.

Лес – множество, содержащее несколько непересекающихся деревьев.

В рассматриваемом алгоритме каждому узлу дерева ставится в соответствие порядковый номер по следующему правилу. Крайнему левому листу присваивается номер 1, данный узел становится помеченным (у помеченных узлов номера не меняются). Проверяются непомеченные узлы у предка первого, крайнему левому присваивается номер 2, следующему 3 и т.д. Номер любого узла всегда больше чем номер его потомка. Корню дерева соответствует номер, равный количеству узлов. Т.е., иначе говоря, нумерация узлов осуществляется в соответствии с так называемым обратным порядком обхода «слева направо» (англ. – «left-to-right postorder»). Пример нумерации узлов дерева представлен на рис. 1.

Операции редактирования, которые рассматриваются в данном алгоритме, – это операции вставки узла, удаления узла и замены метки узла дерева (узел характеризируется номером и меткой). Эти операции представлены графически на рис. 2 («Λ» обозначает некоторый «пустой» («несуществующий») узел).

Данные операции – взвешенные, а весовой коэффициент является метрикой, которая описана в работе [10]. В данном случае, это означает, что вес операции замены метки дерева на такую же метку равен нулю, вес операции вставки метки равен весу удаления этой же метки, вес замены одной метки на другую не может быть больше суммы весов удаления и вставки меток, если эти операции преобразуют исходное дерево к такому же виду, как и в случае замены меток. В данной работе принято считать, что вес какой-либо операции может быть равен только нулю (замена метки на такую же) либо единице (в остальных случаях).

Приняты следующие обозначения.

l(T) – массив номеров листьев дерева T, упорядоченный по возрастанию;

l(T)[i] – номер крайнего левого листа дерева с корнем T[i];

p(T) – массив номеров предков для всех узлов дерева T, упорядоченный по возрастанию;

p(T)[i] – номер предка узла T[i];

LR_keyroots(T) – следующее множество: $\{k \mid \overline{\exists} k' > k : l(T,k) = l(T,k')\}$, т.е. такие k, для которых $l(T,k) \neq l(T,p(T,k))$ (в это множество всегда входит корень дерева T).

Массивы l(T) и LR_keyroots(T) для дерева, представленного на рис. 1, равны соответственно {1, 2, 3, 1, 5, 6, 5, 8, 9, 5, 1} и {2, 3, 6, 8, 9, 10, 11}.

По найденному следу между двумя деревьями можно преобразовать одно дерево к другому. Пример такого преобразования представлен на рис. 3. Метками здесь являются малые символы латинского алфавита.

Приведенная в этой статье реализация алгоритма подразумевает, что каждое из деревьев, в качестве исходных данных, представляется в виде двух массивов: первый – массив предков каждого из узлов дерева: p1 (если это исходное дерево) или p2 (если это конечное дерево), второй – массив меток для каждого из узлов (T1 и T2 соответственно).

Далее будет использована нумерация индексов полей массивов с нуля (нумерация узлов деревьев остается неизменной). В табл. 1 отображены типы нумерации полей параметров деревьев, представленных в виде массивов и списков, которые используются в алгоритме данной статьи, с пояснением смысла индексов и значений полей.

Для решения задачи нахождения расстояния и следа между деревьями алгоритм предусматривает расчет l и LR_keyroots для исходного дерева (l1 и LR_keyroots1 соответственно) и для дерева, к которому нужно преобразовать исходное (l2 и LR_keyroots2 соответственно), следующим образом. Для каждого дерева происходит анализ всех узлов, начиная с первого, в порядке возрастания порядкового номера.

Для примера рассматривается некоторое дерево T. Узлы, которые хотя бы один раз были считаны

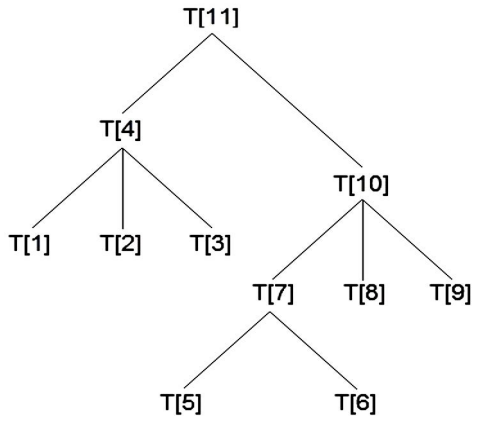

Рис. 1. Дерево и нумерация узлов

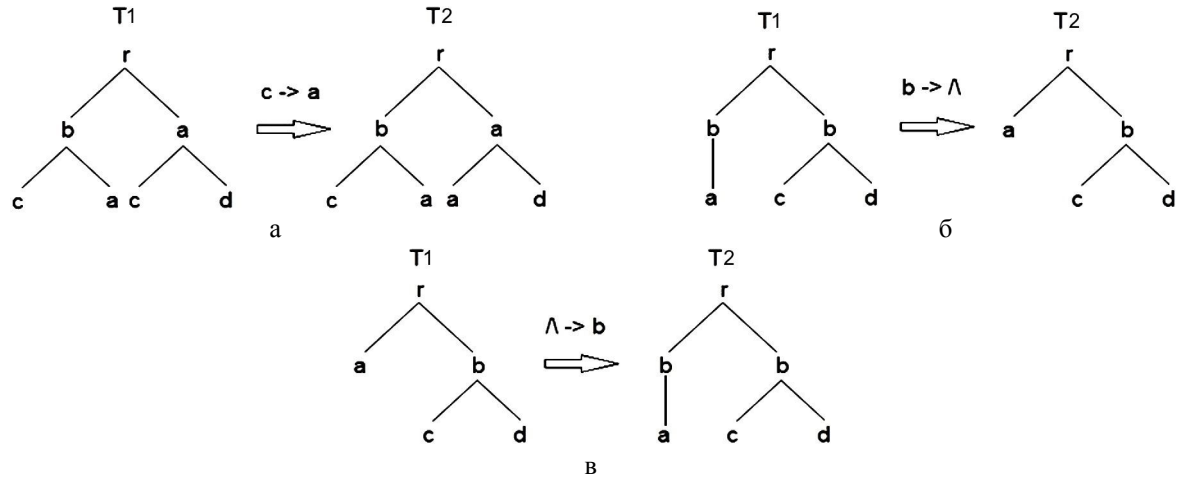

Рис. 2. Операции редактирования: а – замена метки узла a из дерева T1;
б – удаление узла b из дерева T1; в – вставка узла b, находящегося в дереве T2

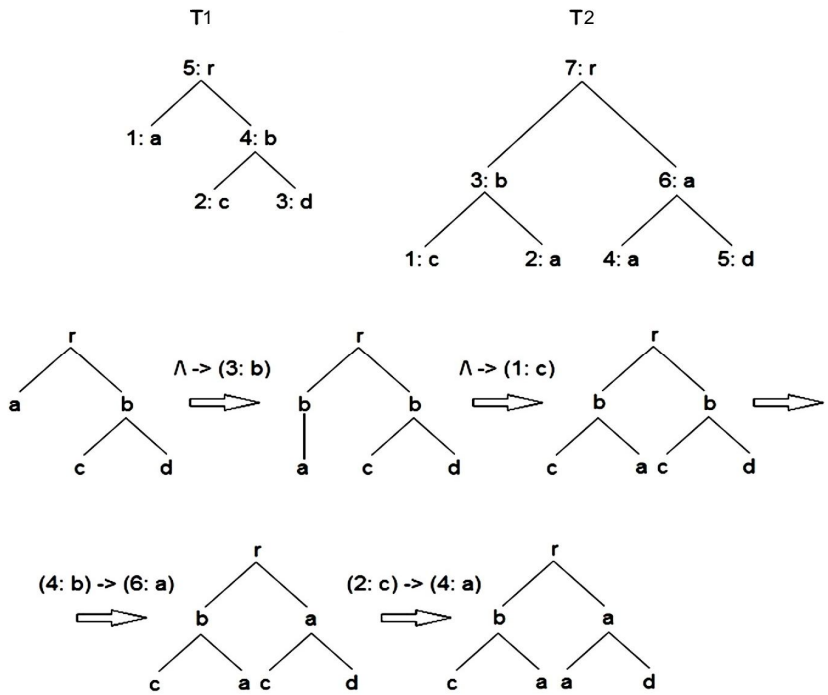

Рис. 3. Процесс изменения дерева

Таблица 1

Форма представления параметров

Параметр	Диапазон нумерации	Индекс	Значение поля
Векторы T1,T2	0..nT1-1, 0..nT2-1 соответственно	номер узла с вычетом единицы	метка дерева
Векторы p1,p2			номер предка
Векторы l1,l2	0..nT1, 0..nT2 соответственно	номер узла (нулевой элемент не используется)	номер крайнего левого листа поддерева
Списки LR_keyroots1, LR_keyroots2	с нуля	номер элемента списка	номер узла согласно определению

или изменены, становятся помеченными. Первоначально массив крайних левых листьев l заполнен нулевыми значениями, а LR_keyroots (как список) пуст.

Очевидно, что

$$\begin{aligned} &l(T)[1] = l(T)[p(T)[1]] = \\ &= l(T)[p(T)[p(T)[1]]] = \ldots = l(T)[|T|] = 1, \end{aligned} \quad (1)$$

где |T| – количество (мощность множества) узлов дерева (в дальнейшем будут использоваться обозначения «nT1» (nT1 = |T1|) и «nT2» (nT2 = |T2|)).

Сначала такими значениями заполняется массив крайних левых листьев l. Далее определяется по порядку следующий непомеченный узел. Если таковой найдется, то это будет лист дерева. Для него справедливо соотношение l(T)[i] = i. В то же время, полю l(T)[p(T)[i]] присваивается значение i, если предок не был помечен; иначе – в массив LR_keyroots добавляется значение i, а затем осуществляется переход к следующему непомеченному узлу. В конце, в массив LR_keyroots добавляется номер корня дерева T.

Распараллеленный алгоритм, рассматриваемый в статье [9], для решения задачи использует два массива:

1) treedist размером nT1*nT2 (treedist[i1, j1] содержит расстояние редактирования между двумя деревьями, корнями которых являются узел T1[i1] дерева T1 и узел T2[j1] дерева T2 соответственно);

2) dist размером $nT1^2*nT2^2$ (dist[i, j][i1, j1] содержит расстояние редактирования между двумя лесами, которые включают в себя узлы с номерами от l1[i] до i1 в дереве T1 и от l2[j] до j1 в дереве T2 соответственно);

Использование массивов таких размеров позволило работать с деревьями с количеством узлов не более двухсот узлов. В связи с этим была осуществлена оптимизация и дополнение взятого за основу распараллеленного алгоритма, предложенного в работе [9], без учета самой параллелизации.

Ниже описаны изменения, которым подвергся этот метод. В работе [9] установлено, что:

$$\begin{cases} \text{dist}[i, j][l1[i]-1, l2[j]-1] = 0, \\ \text{dist}[i, j][i1, l2[j]-1] = \text{dist}[i, j][i1-1, l2[j]-1] + 1, \\ \text{dist}[i, j][l1[i]-1, j1] = \text{dist}[i, j][l1[i]-1, j1-1] + 1, \end{cases} \quad (2)$$
$$\{i1 \in [l1[i]; nT1],\ j1 \in [l2[j]; nT2]\}.$$

Для того чтобы сократить количество полей массива *dist*, в качестве одного из вариантов, необходимо некоторые из них заменить аналитическими выражениями. Следующие из них использованы в данной работе:

$$\begin{cases} \text{dist}[i, j][l1[i]-1, l2[j]-1] = 0, \\ \text{dist}[i, j][i1, l2[j]-1] = i1 - l1[i] + 1, \\ \text{dist}[i, j][l1[i]-1, j1] = j1 - l2[j] + 1, \end{cases} \quad (3)$$
$$\{i1 \in [l1[i]; nT1],\ j1 \in [l2[j]; nT2]\}.$$

Кроме этого, следует заметить, что некоторые поля dist[i, j][i1, j1] массива dist, содержащие расстояние между двумя непустыми лесами, не используются из-за выполнения следующих соотношений:

$$\begin{aligned} &i \in LR\_keyroots1,\ j \in LR\_keyroots2, \\ &i1 \in [l1[i]; i],\ j1 \in [l2[j]; j]. \end{aligned} \quad (4)$$

Пусть исходный массив dist, который будет обозначаться как «dist0», – так называемый «оригинал»; массив dist (отличный от исходного) – «изображение», которое содержит только поля, имеющие суть расстояние между двумя непустыми лесами. Можно заметить, что между индексом поля списка LR_keyroots и значением поля существует взаимно-однозначное отношение. Также можно утверждать, что никогда не произойдет обращения к полю с индексами, которые не удовлетворяют условиям, накладываемым на i1 и j1.

Таким образом, массив dist является рваным, а именно: его можно представить как двумерную матрицу размером |LR_keyroots1|*|LR_keyroots2|, элементы которой – двумерные матрицы, каждая из которых имеет размер, определяемый выражением

$$\begin{aligned} &(LR\_keyroots1[i0] - l1[LR\_keyroots1[i0]] + 1) \times \\ &\times (LR\_keyroots2[j0] - l2[LR\_keyroots2[j0]] + 1), \end{aligned} \quad (5)$$

где i0, j0 – номера строки и столбца первой матрицы соответственно.

Соответствие полей массива dist0 массиву dist можно записать следующим образом:

$$\begin{aligned} &dist0[i, j][i1, j1] \equiv dist[i0, j0][i1\_i, j1\_j], \\ &\{i = LR\_keyroots1[i0], j = LR\_keyroots2[j0], \\ &i1\_i = i1 - ll[i],\ j1\_j = j1 - l2[j]\}. \end{aligned} \quad (6)$$

Также подвергается преобразованию массив treedist. Во-первых, его можно заменить массивом dist0 в соответствии со следующим тождеством:

$$\begin{aligned} &treedist[i1, j1] = dist0[u, v][i1, j1], \\ &\{u \in LR\_keyroots1, ll[u] = ll[i1], \\ &v \in LR\_keyroots2, l2[v] = l2[j1]\}. \end{aligned} \quad (7)$$

Следует заметить, что взаимно-однозначное отношение наблюдается между заданным значением $u \in LR\_keyroots(T)$ и значением $l(T)[u]$.

Пусть задан массив LR_keyroots_inverse(T), для которого справедливо равенство

$$\begin{aligned} &(LR\_keyroots\_inverse(T)[l(T)[i] - 1] = i0) \equiv \\ &\equiv (l(T)[i] = l(T)[i0] \cap i0 \in LR\_keyroots(T)). \end{aligned} \quad (8)$$

Тогда

$$\begin{aligned} &treedist[i1, j1] = dist[LR\_keyroots1\_inverse[ll[i1] - 1], \\ &LR\_keyroots2\_inverse[l2[j1] - 1]][i1 - ll[i1], j1 - l2[j1]], \\ &\{LR\_keyroots1\_inverse = LR\_keyroots\_inverse(T1), \\ &LR\_keyroots2\_inverse = LR\_keyroots\_inverse(T2)\}. \end{aligned} \quad (9)$$

Таким образом, расстояние между деревьями T1 и T2 есть величина, равная

$$\begin{aligned} &dist[|LR\_keyroots1| - 1, \\ &|LR\_keyroots2| - 1][nT1 - 1, nT2 - 1]. \end{aligned} \quad (10)$$

Поиск следа между двумя деревьями осуществляется на основании третьей и пятой лемм статьи [9], используя результат вычисления массива dist.

3. Программная реализация

Ниже представлена реализация алгоритма в виде полного рабочего кода на языке программирования C# (рис. 4).

Считывание данных производится из файла «input.txt», в котором хранится информация о двух деревьях. В первой строке файла содержится значение nT1; в последующих (nT1-1) строках информация представлена в виде двух чисел, разделенных пробелом, суть которых номера узлов дерева T1, при этом первый узел является родителем второго; затем, в nT1 строках представлены метки узлов дерева T1 в виде текста в том порядке, в котором осуществляется нумерация узлов дерева. Аналогичным образом представлена информация о дереве T2 в оставшейся части файла.

4. Анализ программного кода

В табл. 2 представлена информация о примерном времени выполнения программы для выбранных размеров деревьев, при этом размеры обоих деревьев одинаковы, а их глубина равна двум. Тестирование алгоритма проводилось на компьютере с такими характеристиками:

Pentium(R) Dual-Core CPU, 2.2 GHz;

ОЗУ – 3 ГБ;

Тип системы – 32-разрядная ОС.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace Tree_distance_postorder
{
    class Program
    {
        static string[] T1, T2;
        static short nT1, nT2;
        static short[] p1, p2, l1, l2, LR_keyroots1_inverse, LR_keyroots2_inverse;
        static short[,][,] dist;
        static List<short> LR_keyroots1 = new List<short>(),
                           LR_keyroots2 = new List<short>();

        static void Main(string[] args)
        {
            StreamReader sr = new StreamReader("input.txt");
            nT1 = short.Parse(sr.ReadLine());
            p1 = new short[nT1];
```

Рис. 4. Исходный код программы для реализации метода

```
for (short i = 0; i < nT1 - 1; i++)
{
    string s = sr.ReadLine();
    string[] ss = s.Split(' ');
    short a = short.Parse(ss[0]), b = short.Parse(ss[1]);
    p1[b - 1] = a;
}
T1 = new string[nT1];
for (short i = 0; i < nT1; i++)
    T1[i] = sr.ReadLine();
nT2 = short.Parse(sr.ReadLine());
p2 = new short[nT2];
for (short i = 0; i < nT2 - 1; i++)
{
    string s = sr.ReadLine();
    string[] ss = s.Split(' ');
    short a = short.Parse(ss[0]), b = short.Parse(ss[1]);
    p2[b - 1] = a;
}
T2 = new string[nT2];
for (short i = 0; i < nT2; i++)
    T2[i] = sr.ReadLine();
sr.Close();
l1 = new short[nT1 + 1];
l2 = new short[nT2 + 1];
l1[nT1] = l2[nT2] = 1;
LR_keyroots1_inverse = new short[nT1];
LR_keyroots2_inverse = new short[nT2];
for (short i = 1; i < nT1; i++)
{
    for (short j = i; l1[j] == 0 && j < nT1; l1[j] = i, j = p1[j - 1]) ;
    if (l1[p1[i - 1]] != 0 && l1[i] != l1[p1[i - 1]])
    {
        LR_keyroots1_inverse[l1[i] - 1] = (short)LR_keyroots1.Count;
        LR_keyroots1.Add(i);
    }
}
LR_keyroots1_inverse[0] = (short)LR_keyroots1.Count;
LR_keyroots1.Add(nT1);
for (short i = 1; i < nT2; i++)
{
    for (short j = i; l2[j] == 0 && j < nT2; l2[j] = i, j = p2[j - 1]) ;
    if (l2[p2[i - 1]] != 0 && l2[i] != l2[p2[i - 1]])
    {
        LR_keyroots2_inverse[l2[i] - 1] = (short)LR_keyroots2.Count;
        LR_keyroots2.Add(i);
    }
}
LR_keyroots2_inverse[0] = (short)LR_keyroots2.Count;
LR_keyroots2.Add(nT2);
dist = new short[LR_keyroots1.Count, LR_keyroots2.Count][,];
for (short i0 = 0; i0 < LR_keyroots1.Count; i0++)
    for (short j0 = 0; j0 < LR_keyroots2.Count; j0++)
        dist[i0, j0] = new short[
                        LR_keyroots1[i0] - l1[LR_keyroots1[i0]] + 1,
                        LR_keyroots2[j0] - l2[LR_keyroots2[j0]] + 1];
for (short k = 0; k < nT1 + nT2 - 1; k++)
    for (short i0 = 0; i0 < LR_keyroots1.Count; i0++)
        for (short j0 = 0; j0 < LR_keyroots2.Count; j0++)
        {
            short i = LR_keyroots1[i0], j = LR_keyroots2[j0];
            for (short i1 = (short)Math.Max(l1[i], k - j + l1[i] + l2[j]),
                       j1 = (short)(k - i1 + l1[i] + l2[j]);
                       i1 <= i && j1 >= l2[j]; i1++, j1--)
            {
                short loc_delete = l1[i] > i1 - 1 ?
                                   (short)(j1 - l2[j] + 1) :
                                   dist[i0, j0][i1 - l1[i] - 1, j1 - l2[j]];
                short loc_insert = l2[j] > j1 - 1 ?
                                   (short)(i1 - l1[i] + 1) :
                                   dist[i0, j0][i1 - l1[i], j1 - l2[j] - 1];
                short loc_replace;
                if (l1[i] == l1[i1] && l2[j] == l2[j1])
                {
```

Продолжение рис. 4.

```
                            if (l1[i] > i1 - 1 && l2[j] > j1 - 1)
                                loc_replace = 0;
                            else if (l1[i] > i1 - 1)
                                loc_replace = (short)(j1 - l2[j]);
                            else if (l2[j] > j1 - 1)
                                loc_replace = (short)(i1 - l1[i]);
                            else
                                loc_replace =
                                dist[i0, j0][i1 - l1[i] - 1, j1 - l2[j] - 1];
                            dist[i0, j0][i1 - l1[i], j1 - l2[j]] =
                                (short)Math.Min(Math.Min(
                                loc_delete + 1, loc_insert + 1),
                                loc_replace + (T1[i1 - 1] == T2[j1 - 1] ? 0 : 1));
                        }
                        else
                        {
                            if (l1[i] > l1[i1] - 1 && l2[j] > l2[j1] - 1)
                                loc_replace = 0;
                            else if (l1[i] > l1[i1] - 1)
                                loc_replace = (short)(l2[j1] - l2[j]);
                            else if (l2[j] > l2[j1] - 1)
                                loc_replace = (short)(l1[i1] - l1[i]);
                            else loc_replace =
                            dist[i0, j0][l1[i1] - l1[i] - 1, l2[j1] - l2[j] - 1];
                            dist[i0, j0][i1 - l1[i], j1 - l2[j]] =
                            (short)Math.Min(Math.Min(
                            loc_delete + 1,
                            loc_insert + 1),
                            loc_replace +
                            dist[LR_keyroots1_inverse[l1[i1] - 1],
                                 LR_keyroots2_inverse[l2[j1] - 1]][
                                                        i1 - l1[i1], j1 - l2[j1]]);
                        }
                    }
                }
    Console.WriteLine(dist[LR_keyroots1.Count - 1,
                           LR_keyroots2.Count - 1][nT1 - 1, nT2 - 1]);
    Console.WriteLine();
    Console.WriteLine(trace((short)(LR_keyroots1.Count - 1),
                            (short)(LR_keyroots2.Count - 1),
                            (short)(nT1 - 1),
                            (short)(nT2 - 1)));
}

static string trace(short i0, short j0, short i1_i, short j1_j)
{
    string res = "";
    for (; i1_i >= 0 || j1_j >= 0; )
    {
        short i = LR_keyroots1[i0],
              j = LR_keyroots2[j0],
              i1 = (short)(i1_i + l1[i]),
              j1 = (short)(j1_j + l2[j]);
        if (i1_i < 0)
        {
            res += "{};" + T2[j1 - 1] + "-" + j1 + "\r\n";
            j1_j--;
        }
        else
            if (j1_j < 0)
            {
                res += T1[i1 - 1] + "-" + i1 + ";{}\r\n";
                i1_i--;
            }
            else
            {
                short loc_insert = j1_j < 1 ?
                                   (short)(i1_i + 1) :
                                   dist[i0, j0][i1_i, j1_j - 1],
                      loc_replace;
                if (l1[i] == l1[i1] && l2[j] == l2[j1])
                {
                    if (i1_i < 1 && j1_j < 1)
                        loc_replace = 0;
```

Продолжение рис. 4.

```
                        else if (i1_i < 1)
                            loc_replace = j1_j;
                        else if (j1_j < 1)
                            loc_replace = i1_i;
                        else
                            loc_replace = dist[i0, j0][i1_i - 1, j1_j - 1];
                        if (dist[i0, j0][i1_i, j1_j] ==
                            loc_replace + (T1[i1 - 1] == T2[j1 - 1] ? 0 : 1))
                        {
                            res += T1[i1 - 1] + "-" + i1 + ";" +
                                   T2[j1 - 1] + "-" + j1 + "\r\n";
                            i1_i--;
                            j1_j--;
                        }
                        else
                            if (dist[i0, j0][i1_i, j1_j] == loc_insert + 1)
                            {
                                res += "{};" + T2[j1 - 1] + "-" + j1 + "\r\n";
                                j1_j--;
                            }
                            else
                            {
                                res += T1[i1 - 1] + "-" + i1 + ";{}\r\n";
                                i1_i--;
                            }
                    }
                    else
                    {
                        if (l1[i] > l1[i1] - 1 && l2[j] > l2[j1] - 1)
                            loc_replace = 0;
                        else if (l1[i] > l1[i1] - 1)
                            loc_replace = (short)(l2[j1] - l2[j]);
                        else if (l2[j] > l2[j1] - 1)
                            loc_replace = (short)(l1[i1] - l1[i]);
                        else
                            loc_replace =
                            dist[i0, j0][l1[i1] - l1[i] - 1, l2[j1] - l2[j] - 1];
                        if (dist[i0, j0][i1_i, j1_j] ==
                            loc_replace +
                            dist[LR_keyroots1_inverse[l1[i1] - 1],
                                 LR_keyroots2_inverse[l2[j1] - 1]][
                                                        i1 - l1[i1], j1 - l2[j1]])
                        {
                            res += trace(i0, j0,
                                   (short)(l1[i1] - l1[i] - 1),
                                   (short)(l2[j1] - l2[j] - 1));
                            i0 = LR_keyroots1_inverse[l1[i1] - 1];
                            i1_i = (short)(i1 - l1[i1]);
                            j0 = LR_keyroots2_inverse[l2[j1] - 1];
                            j1_j = (short)(j1 - l2[j1]);
                        }
                        else
                            if (dist[i0, j0][i1_i, j1_j] == loc_insert + 1)
                            {
                                res += "{};" + T2[j1 - 1] + "-" + j1 + "\r\n";
                                j1_j--;
                            }
                            else
                            {
                                res += T1[i1 - 1] + "-" + i1 + ";{}\r\n";
                                i1_i--;
                            }
                    }
                }
            }
            return res;
        }
    }
}
```

Окончание рис. 4.

Алгоритм Шаши-Жанга имеет подобные характеристики, за исключением того, что его работоспособность сохраняется до ста узлов в обоих деревьях.

Заключение

Рассмотренный алгоритм является модификацией алгоритма Шаши-Жанга в том смысле, что

последний не позволяет определить след между двумя деревьями.

Таблица 2

Данные о работе программы

Кол-во узлов каждого дерева	Примерное время выполнения		Исключение
	поиск расстояния	поиск следа	
10	меньше 1 с	меньше 1 с	–
50	1 с	1 с	
100	1-2 с	1 с	
200	2 с	2 с	
500	2 с	2 с	
1000	1 мин	5 с	
2000	20 мин	5 с	
4000	2 ч	–	StackOverFlow
7000	–	–	OutOfMemory

Описанный в работе алгоритм обладает той особенностью, что при числе узлов порядка нескольких тысяч сохраняет свою работоспособность, т.к. в нем используемая память растет медленнее, чем в алгоритме Шаши-Жанга, при увеличении числа узлов деревьев. Если использовать в качестве меток строки большой длины, программа может завершиться с сообщением о недостаточном объеме памяти.

Планируется создание обучающей системы, которая будет анализировать написанный пользователем код – проверять его на множестве тестов.

В случае, если пользователь не может решить поставленную задачу, система начнет лексический анализ кода (сравнение массивов лексем), а затем синтаксический (сравнение абстрактных синтаксических деревьев), для того чтобы выявить место ошибки и при помощи подсказок приблизить обучаемого к правильному написанию алгоритма.

Литература

1. Identifying and summarizing systematic code changes via rule inference [Text] / M. Kim, D. Notkin, D. Grossman, G. Wilson Jr. // IEEE Transactions on Software Engineering – 2013. – Vol. 39(1). – P. 45–62.

2. Rattan, D. Software clone detection: A systematic review [Text] / D. Rattan, R. Bhatia, M. Singh // Information and Software Technology – 2013. – Vol. 55(7). – P. 1165 – 1199.

3. Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments [Text] / M. Vujošević-Janičić, M. Nikolić, D. Tošić, V. Kuncak // Information And Software Technology. – 2013. – Vol. 55(6). – P. 1004–1016.

4. Increasing Adoption of Smart Learning Content for Computer Science Education [Text] / P. Brusilovsky, S. Edwards, A. Kumar, L. Malmi, L. Benotti, D. Buck, P. Ihantola, R. Prince, T. Sirki¨a, S. Sosnovsky, J. Urquiza, A. Vihavainen, M. Wollowski // In Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference. – New York, 2014. – P. 31–57.

5. An Intelligent Tutoring System for Learning Java Objects [Text] / S. Abu-Naser, A. Ahmed, N. Al-Masri, A. Deeb, E. Moshtaha, M. AbuLamdy // International Journal of Artificial Intelligence & Applications (IJAIA). – 2011. – Vol. 2(2). – P. 68 – 77.

6. Butz, C. Web-based intelligent tutoring system for computer programming [Text] / C. Butz, S. Hua, R. Maguire // Web Intelligence and Agent Systems: An International Journal. – 2006. – Vol. 4(1). – P. 77 – 97.

7. Sykes, E. R. A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java [Text] / E. R. Sykes, F. Franek // International Conference on Computers and Advanced Technology in Education, Rhodes, Greece, June 30–July 2, 2003. – Rhodes, Greece, 2003. – P. 78 – 83.

8. Striewe, M. A Review of Static Analysis Approaches for Programming Exercise [Text] / M. Striewe, M. Goedicke // Communications in Computer and Information Science. – 2014. – Vol. 439. – P. 100 – 113.

9. Zhang, K. Simple fast algorithms for the editing distance between trees and related problems [Text] / K. Zhang, D. Shasha // Society for Industrial and Applied Mathematics. Journal on Computing. – 1989. – Vol. 18(6). – P. 1245 – 1262.

10. Tai, K.C. The tree-to-tree correction problem [Text] / K.C. Tai // Journal of the ACM. – 1979. – Vol. 26 (3). – P. 422–433.

References

1. Kim, M., Notkin, D., Grossman, D., Wilson, G. Jr. Identifying and summarizing systematic code changes via rule inference. *IEEE Transactions on Software Engineering*, 2013, vol. 39, no. 1, pp. 45–62. doi: 10.1109/TSE.2012.16.

2. Rattan, D., Bhatia, R., Singh, M. Software clone detection: A systematic review. *Information and Software Technology*, 2013, vol. 55, no. 7, pp. 1165 – 1199. doi:10.1016/j.infsof.2013.01.008.

3. Vujošević-Janičić, M., Nikolić, M., Tošić, D., Kuncak, V. Software Verification and Graph Similarity for Automated Evaluation of Students' Assignments. *Information And Software Technology*, 2013, vol. 55, no 6, pp. 1004–1016. doi:10.1016/j.infsof.2012.12.005.

4. Brusilovsky, P., Edwards, S., Kumar, A., Malmi, L., Benotti, L., Buck, D., Ihantola, P., Prince, R., Sirki¨a, T., Sosnovsky, S., Urquiza, J., Vihavainen, A., Wollowski, M. Increasing Adoption of Smart Learning Content for Computer Science Education. *In Proceedings of the Working Group Reports of the 2014 on Inno-*

vation & Technology in Computer Science Education Conference. New York, 2014, pp. 31–57. doi: 10.1145/2713609.2713611.

5. Abu-Naser, S., Ahmed, A., Al-Masri, N., Deeb, A., Moshtaha, E., AbuLamdy, M. An Intelligent Tutoring System for Learning Java Objects. *International Journal of Artificial Intelligence & Applications (IJAIA)*, 2011, vol. 2, no. 2, pp. 68 – 77. doi: 10.5121/ijaia.2011.2205.

6. Butz, C., Hua, S., Maguire, R. Web-based intelligent tutoring system for computer programming. *Web Intelligence and Agent Systems: An International Journal*, 2006, vol. 4, no. 1, pp. 77 – 97. doi: 10.1109/WI.2004.10104.

7. Sykes, E. R., Franek, F. A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java. *International Conference on Computers and Advanced Technology in Education*. Rhodes, Greece, 2003, pp. 78 – 83. doi: 10.2316/Journal.208.2004.1.202-1454.

8. Striewe, M., Goedicke, M. A Review of Static Analysis Approaches for Programming Exercise. *Communications in Computer and Information Science*, 2014, vol. 439, pp. 100 – 113. doi: 10.1007/978-3-319-08657-6_10.

9. Zhang, K., Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. *Society for Industrial and Applied Mathematics. Journal on Computing*, 1989, vol. 18, no. 6, pp. 1245 – 1262. doi: 10.1137/0218082.

10. Tai, K.C. The tree-to-tree correction problem. *Journal of the ACM*, 1979, vol. 26, no. 3, pp. 422–433. doi: 10.1145/322139.322143.

Поступила в редакцию 03.02.2016, рассмотрена на редколлегии 18.02.2016

МЕТОД СТРУКТУРНОГО ДІАГНОСТУВАННЯ СТУДЕНТСЬКИХ КОМП'ЮТЕРНИХ ПРОГРАМ

Д. О. Гайдачук, А. Г. Чухрай

Дана робота присвячена дослідженню питання навчання студентів за допомогою інтелектуальних комп'ютерних програм, що навчають (ІКПН) у галузі алгоритмізації та програмування. Розглядається одне з питань, які виникають при навчанні професіональним умінням алгоритмізації та програмування, а саме – як порівняти еталонну програму, що зберігається в ІКПН, із програмою, яка складена тим, кого навчають. Пропонується метод структурного діагностування комп'ютерних програм, розроблених студентами, який має за основу модифікацію існуючого алгоритму порівняння двох структур даних, а саме m-арних дерев. Такими деревами можуть бути абстрактні синтаксичні дерева.

Ключові слова: інтелектуальна комп'ютерна програма, що навчає, m-арне дерево, відстань редагування, слід, програма, розроблена студентом, еталонна програма.

METHOD OF STUDENTS COMPUTER PROGRAMS STRUCTURAL DIAGNOSING

D. A. Haidachuk, A. G. Chukhray

This work is dedicated to studying student problem research with Intelligent Tutoring Systems in area of algorithmization and programming. One of the issues that arises while teaching for professional skills in algorithmization and programming is considered, specifically is how to compare reference program stored in ITS with the program developed by a student. Structural diagnosis method for computer programs developed by students is proposed. This method is based on modification of existing method for comparing two data structures, specifically are m-ary trees which can be as an abstract syntax trees.

Key words: Intelligent Tutoring System; m-ary tree; editing distance; trace; program developed by student; reference program.

Гайдачук Даниэль Александрович – студент кафедры систем управления летательными аппаратами, Национальный аэрокосмический университет им. Н. Е. Жуковского «Харьковский авиационный институт», Харьков, Украина, e-mail: dvrch@mail.ru.

Чухрай Андрей Григорьевич – д-р техн. наук, доцент, профессор кафедры систем управления летательными аппаратами, Национальный аэрокосмический университет им. Н. Е. Жуковского «Харьковский авиационный институт», Харьков, Украина, e-mail: achukhray@gmail.com.

Haidachuk Daniel Aleksandrovich – Student of Aircraft Control Systems Department, Zhukovskyi National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine, e-mail: dvrch@mail.ru.

Chukhray Andrey Grigorevich – Doctor of Technical Sciences, Professor of Aircraft Control Systems Department, Zhukovskyi National Aerospace University "Kharkiv Aviation Institute", Kharkov, Ukraine, e-mail: achukhray@gmail.com.