ЗАДАЧА СРАВНЕНИЯ ИЕРАРХИЧЕСКИХ МОДЕЛЕЙ ДАННЫХ. ПОДХОДЫ К РЕШЕНИЮ

А. В. Лычковский

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники», г. Минск

Научный руководитель Н. А. Волорова

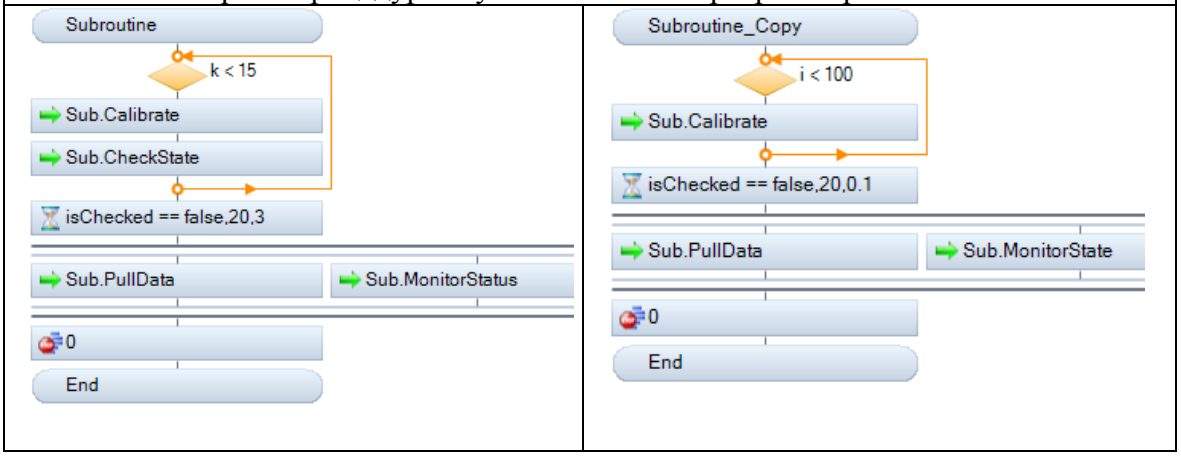
Существует много областей, где сравнение данных является частью процесса работы. Рассмотрим некоторые примеры информации, которая может нуждаться в сравнении по тем или иным причинам.

Часто возникает задача конфигурирования программного обеспечения сходным образом, но с сохранением некоторых персональных параметров конфигурации. Также может возникнуть задача нахождения различия в конфигурации программного обеспечения для устранения неправильной работы или в других целях.

Информация, которая требует сравнения, не всегда представлена в виде текста. Часто информация визуализируется и для ее редактирования используется специальное программное обеспечение. Сложность анализа информации, представленной в таком виде, заключается в том, что для получения полной информации об элементах часто приходится пользоваться их редакторами, так как не все детали могут быть отображены из-за небольшого пространства.

Если проанализировать структуру информации представленных примеров, то можно заметить наличие иерархии. Информация может быть разбита на уровни. При таком разбиении каждый уровень будет представлен набором элементов, каждый из которых также может быть разбит на уровни. Информация, разбитая на уровни указанным способом, может быть представлена иерархической моделью данных.

Разные версии процедур визуального языка программирования



Использование модели данных для представления информации дает некоторые преимущества:

– абстрагирование от источника информации;

– абстрагирование от способа представления информации;

– использование информации разной по структуре.

Разработка эффективного метода сравнения таких моделей данных позволит решать прикладные задачи быстрее и эффективнее.

В результате поиска в сети Интернет были найдены программы, позволяющие сравнивать файлы с данными разных форматов, файловые директории. Также были обнаружены онлайн-сервисы, позволяющие сравнивать продукты, продаваемые онлайн-магазинами. Файлы данных и продукты онлайн-магазинов могут быть представлены простой иерархической моделью данных, которая имеет один уровень. Для представления структуры файловой директории понадобятся дополнительные уровни.

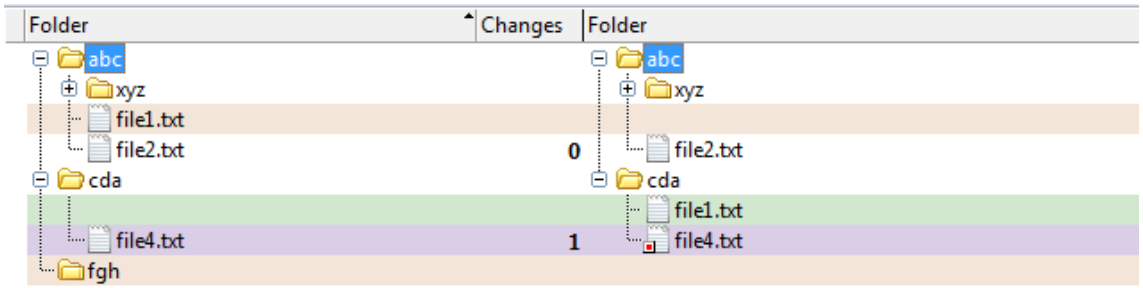


Рис. 1. Пример результата сравнения файловых директорий одной из найденных программ

Сравнение иерархических моделей данных можно свести к сравнению двух последовательностей элементов. Сравнение должно начинаться с верхнего уровня, а при сравнении элементов этого уровня необходимо учитывать, что каждый из них может также иметь иерархическую структуру. В этом случае каждый из таких элементов сравнивается как самостоятельный элемент. То есть мы получаем рекурсивный алгоритм сравнения, который будет продолжаться, пока сравниваемый элемент будет иметь дочерние элементы в своей структуре.

Для оптимальной работы алгоритма, описанного выше, необходимо решить задачу сравнения последовательностей элементов. В теории алгоритмов эта задача из-

вестна как задача о нахождении наибольшей общей подпоследовательности (англ. longest common subsequence, LCS).

Полный перебор. Существуют разные подходы при решении данной задачи при полном переборе – можно перебирать варианты подпоследовательности, варианты вычеркивания из данных последовательностей и т. д. Однако в любом случае, время работы программы будет экспонентой от длины строки.

Метод динамического программирования. Данная задача может быть решена методом динамического программирования. Вначале найдем длину наибольшей подпоследовательности. Допустим, мы ищем решение для случая (n_1, n_2), где n_1, n_2 – длины первой и второй последовательностей. Пусть уже существуют решения для всех подзадач (m_1, m_2), меньших заданной, тогда задача (n_1, n_2) сводится к меньшим подзадачам следующим образом:

$$f(n_1,n_2)=\begin{cases} 0, & n_1=0 \vee n_2=0 \\ f(n_1-1,n_2-1)+1, & s[n_1]=s[n_2] \\ \max(f(n_1-1,n_2),f(n_1,n_2-1)), & s[n_1]\neq s[n_2] \end{cases}$$

Теперь вернемся к задаче построения подпоследовательности. Для этого в существующий алгоритм добавим запоминание для каждой задачи той подзадачи, через которую она решается. Следующим действием, начиная с последнего элемента, поднимаемся к началу по направлениям, заданным первым алгоритмом, и записываем символы в каждой позиции. Это и будет ответом в данной задаче. Время работы алгоритма будет $0(n_1 \cdot n_2)$.

Расстояние Левенштейна. Расстояние Левенштейна (также редакционное расстояние или дистанция) между двумя строками в теории информации и компьютерной лингвистике – это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Пусть S_1 и S_2 – две строки (длиной M и N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) можно подсчитать по следующей рекуррентной формуле:

$$d(S_1,S_2)=D(M,N),$$

где $$D(i,j)=\begin{cases} 0, & i=0,j=0 \\ i, & j=0,i>0 \\ j, & i=0,j>0 \\ \min(D(i,j-1)+1,D(1-1,j)+1,D(i-1,j-1)+m(S_1[i],S_2[i])), & j=0,i>0 \end{cases}$$

где функция $m(a, b)$ равна нулю, если $a = b$ и единице в противном случае.

Расстояние Левенштейна может быть обобщено на последовательности элементов. Также можно использовать разные цены для операций вставки, удаления, замены символа.

Задача вычисления расстояния Левенштейна может быть решена методом динамического программирования, при этом время работы алгоритма будет $O(M \cdot N)$.

Расстояние Дамерау–Левенштейна. Если к списку разрешенных операций добавить транспозицию (два соседних символа меняются местами), получается расстояние Дамерау–Левенштейна. Для нее также существует алгоритм, требующий

$O(M \cdot N)$ операций. Дамерау показал, что 80 % ошибок при наборе текста человеком являются транспозициями. Кроме того, расстояние Дамерау–Левенштейна используется и в биоинформатике.

An O(ND) Difference Algorithm. Этот алгоритм позволяет находить наибольшую общую подпоследовательность строк и требует $O(N \cdot D)$ операций, где N – сумма длин сравниваемых строк, а D – размер минимального редакционного предписания. Алгоритм эффективен, когда сравниваемые сроки очень сходны. В своей работе алгоритм использует графы для решения задачи. Этот алгоритм также можно обобщить для сравнения последовательностей элементов.

В результате проделанной работы были исследованы примеры из реальной жизни, которые делают задачу сравнения иерархических моделей данных актуальной. Были проанализированы существующие решения поставленной задачи. Также были найдены алгоритмы и структуры данных, позволяющие решать поставленную задачу. На основании этих алгоритмов может быть разработано программное обеспечение, которое решает задачу сравнения с учетом индивидуальных особенностей иерархических моделей.

Литература

1. Longest common subsequence problem [Электронный ресурс]. – Электронные данные. – Режим доступа: http://en.wikipedia.org/wiki/Longest_common_subsequence_problem.
2. Levenshtein distance [Электронный ресурс]. – Электронные данные. – Режим доступа: http://en.wikipedia.org/wiki/Levenshtein_distance.
3. Damerau–Levenshtein distance [Электронный ресурс]. – Электронные данные. – Режим доступа: http://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance.
4. An O(ND) Difference Algorithm and Its Variations/ Eugene W. Myers [Электронный ресурс]. – Электронные данные. – Режим доступа: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.6927.