АВТОМАТИЗАЦИЯ СОСТАВЛЕНИЯ ВАРИАНТОВ ЗАДАНИЙ ДЛЯ ПРОВЕРОЧНЫХ РАБОТ

П.Ю. Маврин, В.Г. Парфенов, А.С. Станкевич
Санкт-Петербургский государственный университет информационных технологий, механики и оптики
Тел.: (812) 232-46-20, e-mail: pavel.mavrin@gmail.com

При обучении важно своевременно проверять знания учащихся. Проведение частых проверочных (контрольных, лабораторных и т.д.) работ полезно для всех участников образовательного процесса:

- Для учеников – побуждает повторить и структурировать материал при подготовке к проверке.
- Для преподавателей – помогает понять, насколько хорошо учащиеся воспринимают материал.
- Для руководителей учебных организаций – помогает вести мониторинг учебной работы.

Основная проблема, с которой сталкиваются преподаватели при проведении проверочных работ – ученики списывают ответы друг у друга. Следить за этим, как правило, сложно (особенно если работа выполняется дома), а определить по работе, что она списана, порой невозможно (например, в случае проведения тестирования с выбором из нескольких вариантов).

Самый распространенный способ борьбы со списыванием – составление нескольких вариантов заданий. При этом ученики, получившие разные варианты заданий, не могут списать решения друг у друга. Данная система применяется повсеместно и хорошо себя зарекомендовала.

Однако у такой системы есть большой недостаток. Объем работы, который необходимо проделать преподавателю увеличивается пропорционально количеству вариантов. Поэтому, как правило, составляется не более пяти вариантов задания. Такого количества вариантов обычно бывает достаточно, если проверочная работа проводится один раз, одновременно для всех учеников и под присмотром преподавателя. Если хотя бы одно из перечисленных условий нарушено, то ученики могут достаточно быстро составить список решений всех вариантов и распространить его между собой.

В данной работе предлагается решить эту проблему при помощи автоматической генерации вариантов задания. При таком подходе можно построить неограниченное (строго говоря, ограниченное, но достаточно большое) число вариантов, что позволит избежать проблем, описанных выше.

Пусть требуется составить задания по переводу чисел из одной системы счисления в другую. Тогда можно, например, составить такой генератор:

```
<%@ page import="static util.Random " %>
Переведите число <%=randomInt(0x100, 0x1000)%>
в шестнадцатеричную систему счисления.
```

В данном коде используется метод **`int randomInt(int min, int max)`**, возвращающий случайное целое число из интервала от **`min`** (включительно) до **`max`** (не включительно). Данный метод описан в классе **`util.Random`** (код не приводим), который импортируется в первой строке генератора.

Запустив приведенный генератор и передав ему в качестве параметра значение зерна, получим на выходе конкретный вариант задания. Например:

```
Переведите число 1465 в шестнадцатеричную систему счисления.
```

Границы подобраны таким образом, чтобы ответы в получающихся вариантах всегда были трехзначными, таким образом, сложность всех сгенерированных вариантов будет примерно одинаковой.

Рассмотрим различные дополнительные интересные свойства, которые можно придать автоматически сгенерированным вариантам задания.

К вариантам заданий, предлагаемых на проверочную работу, предъявляется ряд требований. Перечислим некоторые из них:

- Вариант должен проверять знания ученика по теме проверочной работы.
- Вариант должен быть не слишком простым.
- Вариант должен быть не слишком сложным.
- У ученика не должно быть возможности случайно угадать правильный ответ.

Как составлять варианты удовлетворяющие требованиям? Как в известном анекдоте есть два метода: «подумать» и «потрясти».

Метод 1, «подумать». Перед тем, как писать генератор, проектируется общий вид качественного (удовлетворяющего требованиям) варианта. Далее составляется генератор таким образом, чтобы получались только такие варианты.

Метод 2, «потрясти». Составляется простой генератор вариантов, не учитывающий качество генерируемых вариантов, и программа-анализатор, составляющая оценку варианта с точки зрения предъявляемых требований. Далее для того, чтобы сгенерировать хороший вариант, простой генератор запускается несколько раз до тех пор, пока анализатор не признает сгенерированный вариант качественным.

Метод «потрясти» отличается большей универсальностью, так как проверить удовлетворение свойств, как правило, проще, чем построить объект, им удовлетворяющий. Однако применение этого метода «в лоб» часто оказывается невозможным (например, если вероятность случайно получить хороший вариант крайне низка) или нецелесообразным (когда проще применить метод «подумать»). Иногда также бывает полезно

использовать комбинацию этих методов (запускать «умный» генератор несколько раз, анализируя качество получаемых вариантов).

Для удобства проверки заданий преподавателю требуется предоставить правильный ответ (или набор правильных ответов). Для того чтобы сгенерировать ответ к заданию, можно построить программу-решатель, которая будет получать на вход текст задания и выдавать текст ответа. Такая схема удобна, например, для заданий вида «Что выведет данная программа?». В этом случае для получения правильного ответа программу надо просто скомпилировать и запустить. В других случаях построить такой решатель может быть затруднительно и удобнее использовать схему двухэтапной генерации. На первом этапе по зерну генерируются набор параметров варианта. На втором этапе из этих данных получаются два файла – задание и ответ на него. В этом случае благодаря предварительному этапу решаются две проблемы. Во-первых, решателю не требуется анализировать текст задания, а во-вторых, при надобности в параметры варианта можно включить дополнительные «подсказки» для его решения.

Предложенный метод – действенный способ борьбы со списыванием на проверочных работах. Он был апробирован при проведении контрольных и лабораторных работ для студентов СПбГУ ИТМО и показал хорошие результаты. Метод применялся при составлении задач по дискретной математике, информатике, программированию, однако он может быть применен и в других областях, например в физике.

Основное достоинство метода – автоматизация создания большого количества вариантов заданий и правильных ответов на них.

Один из главных недостатков метода – обязательное участие программиста в процессе составления генератора заданий. Как показала практика, использование языка JSP сильно упрощает программистскую составляющую процесса построения генератора, однако для дальнейшего развития метода необходимо создание специализированного языка, упрощающего разработку генераторов, что позволит создавать генераторы людям, не являющихся профессиональными программистами (например, учителям математики и физики).