ПЕДАГОГИЧЕСКИЕ ИЗМЕРЕНИЯ И ТЕСТЫ

В. В. Кручинин,
канд. техн. наук, зам. директора по научной работе Томского межвузовского центра дистанционного образования, зав. лабораторией инструментальных систем моделирования и обучения Томского государственного университета систем управления и радиоэлектроники

МЕТОДЫ ГЕНЕРАЦИИ ТЕСТОВЫХ ЗАДАНИЙ ПО ИНФОРМАТИКЕ

В современных условиях компьютерные учебные программы становятся необходимым атрибутом процесса обучения [1]. Одним из важных элементов подобных программ являются *генераторы*, которые обеспечивают конструирование вопросов, задач и заданий [2, 3]. Схемы использования генераторов могут быть различными: построение индивидуальных заданий для проведения практических занятий, синтез вопросов для организации контроля и самоконтроля, конструирование задач в различных тренажерах и др.

В основе генераторов лежат *модели и алгоритмы синтеза вопросов и задач*. Эти модели могут иметь универсальный характер или зависеть от предметной области [2—6]. Ниже рассматриваются модели и алгоритмы для генерации вопросов и задач *применительно к информатике*. Анализируются следующие методы генерации вопросов:

- для циклических алгоритмов;
- на основе словарей;
- на основе использования деревьев И-ИЛИ.

Записаны обобщенные методики построения алгоритмов генерации, приведены конкретные примеры.

Генерация вопросов на основе алгоритмов

Алгоритм — одно из базовых понятий информатики. Многие алгоритмы имеют циклический характер, например вычисление суммы ряда, сортировка массивов, обработка матриц и т. д. Анализ таких алгоритмов приводит к пониманию, что можно предложить единую методику генерации вопросов и тестовых заданий для алгоритмов подобного класса.

Рассмотрим простейший пример алгоритма нахождения суммы натурального ряда:

Шаг 1: $i = 1, S = 0$.
Шаг 2: $S = S + i$.
Шаг 3: $i = i + 1$.
Шаг 4: Если $(i < n)$, то перейти к шагу 2.

Можно предложить учащемуся это описание и задать следующий вопрос:

Какое значение примет переменная S после завершения цикла, если n равно 5?

При создании компьютерного теста данное задание будет выглядеть (в закрытой форме) следующим образом:

Дан следующий алгоритм:

Шаг 1: $i = 1, S = 0$.
Шаг 2: $S = S + i$.
Шаг 3: $i = i + 1$.
Шаг 4: Если $(i < n)$, то перейти к шагу 2.

Какое значение примет переменная S после завершения цикла, если n равно 5?

Введите найденное значение S:

После ответа, введенного учеником, может быть выведен правильный ответ, например так:

Правильный ответ: 10.

В открытой форме задание может быть сформулировано следующим образом:

Дан следующий алгоритм:

Шаг 1: $i = 1, S = 0$.
Шаг 2: $S = S + i$.
Шаг 3: $i = i + 1$.
Шаг 4: Если $(i < n)$, то перейти к шагу 2.

Какое значение примет переменная S после завершения цикла, если n равно 5?

Укажите правильный вариант:

1) $S = 20$
2) $S = 10$
3) $S = 6$
4) $S = 15$

После ответа, введенного учащимся, выводится правильный ответ:

Правильный вариант — 2.

Тест, составленный из таких вопросов, при его многократном использовании приводит к эффекту, когда учащийся механически запоминает правильный ответ и уже без должного анализа отвечает на поставленный вопрос.

Для решения данной проблемы предлагается создать программу, которая обеспечивает создание конкретного вопроса из параметризированного описания вопроса. Тогда ***для генерации вопроса необходимо***:

1) записать вопрос в параметризированном виде;
2) записать условия установки начальных значений параметров;
3) записать алгоритм поиска правильного решения;
4) записать алгоритм формулировки вопроса.

Рассмотрим построение такой программы для нашего примера.

1. *Параметризированный вопрос:*

Дан следующий алгоритм:

Шаг 1: $i = 1, S = 0$.
Шаг 2: $S = S + i$.
Шаг 3: $i = i + 1$.
Шаг 4: Если $(i < n)$, то перейти к шагу 2.

Какое значение примет переменная S после завершения цикла, если n равно k?

Введите найденное значение S:

Здесь параметром является k — ограничение цикла на шаге 3 приведенного алгоритма.

2. *Условие установки начальных значений:* параметр k случайно выбрать из диапазона целых чисел [2, 8].

3. *Алгоритм поиска правильного решения:* написать программу нахождения суммы натурального ряда, заданного параметром k.

4. *Формулировка конкретного вопроса:* поставить значение k в текст параметризированного вопроса, правильный ответ найти, вызвав подпрограмму — функцию нахождения суммы. Для записи вопроса в форме меню можно получить значения суммы при других значения параметра k.

Очевидно, что количество вопросов будет равно количеству различных значений параметра, а это определяется границами заданного интервала (в нашем случае —

7 вариантов). Изменяя значение границ, можно получить большее число вопросов. При такой организации формирования тестовых вопросов при повторном тестировании учащемуся может выпасть этот вопрос, но с другим параметром k. И, соответственно, с другим правильным ответом. Поэтому для ответа на вопрос учащемуся необходимо изучить алгоритм, а не запомнить конкретное число.

Далее будем называть параметризированный вопрос *шаблоном вопроса*.

Рассмотрим **основные типы шаблонов для алгоритмов, имеющих циклы.**

Обозначим все переменные, которые изменяются в цикле, как $\{s_j\}_{j=1}^{n} \cup i$, где i — переменная, задающая номер итерации, s_j — другие переменные, изменяющиеся в цикле, n — количество переменных.

Тогда можно записать в общем случае следующие шаблоны:

- Известны начальные значения переменных $\{s_j\}_{j=1}^{n}$, необходимо найти конечные значения переменных $\{s_j\}_{j=1}^{n}$.
- Известны начальные значения переменных $\{s_j\}_{j=1}^{n}$, необходимо найти значения переменных $\{s_j\}_{j=1}^{n}$ на i-й итерации.
- Известны конечные значения переменных $\{s_j\}_{j=1}^{n}$, необходимо найти начальные значения переменных $\{s_j\}_{j=1}^{n}$.
- Известны конечные значения переменных $\{s_j\}_{j=1}^{n}$, необходимо найти промежуточные значения переменных $\{s_j\}_{j=1}^{n}$ на i-й итерации.
- Известны начальные и промежуточные значения переменных $\{s_j\}_{j=1}^{n}$, необходимо найти номер итерации цикла.
- Известны начальные и конечные значения переменных $\{s_j\}_{j=1}^{n}$, необходимо найти количество итераций цикла.

Для рассматриваемого нами примера можно записать следующие конкретные шаблоны (с учетом того, что один шаблон рассмотрен выше и здесь мы его рассматривать не будем):

1. Какое значение примет переменная S на итерации $i = k$?
2. Какое значение переменной n было установлено, если по завершении цикла значение $S = k$?
3. Если в шаге 1 начальное значение $i = 1$ заменить на параметр ($i = i_0$), то можно задать следующий вопрос: «Какое начальное значение должна иметь переменная i, если $S = k$?»
4. На какой итерации значение S станет равным k?
5. Сколько итераций было выполнено, если по завершении цикла значение $S = k$?

Для построения генератора необходимо для каждого из шаблонов задать алгоритм генерации параметров и алгоритм нахождения правильного ответа. Казалось бы, для каждого шаблона необходимо писать свои алгоритмы генерации и нахождения правильного ответа. Однако можно получить более экономное решение. Запишем основные параметры алгоритма:

n — количество итераций цикла,
S — искомая сумма,
i — номер текущей итерации,
i_0 — начальное значение переменной i.

Алгоритм нахождения суммы натурального ряда обозначим как $FuncSum(n, i_0)$.

Обозначим алгоритм генерации параметров как $Gen(a, b)$, где
a — нижняя граница изменения;
b — верхняя граница изменения.

Запишем теперь **алгоритмы генерации для перечисленных шаблонов:**

1. Какое значение примет переменная S на итерации $i = k$?
$n = Gen(a, b)$

$i = Gen(a, b)$
$sum = FuncSum(i, 0)$
Полученное значение i подставляется вместо параметра k, в правильный ответ записывается значение sum. Параметры n, i генерируются из условия $n - i > 0$.

2. Какое значение переменной n было установлено, если по завершении цикла значение $S = k$?
$n = Gen(a, b)$
$sum = FuncSum(n, 0)$
Полученное значение sum подставляется вместо параметра k, в правильный ответ записывается значение n.

3. Какое начальное значение должна иметь переменная i, если $S = k$?
$n = Gen(a, b)$
$i_0 = Gen(a, b)$
$sum = FuncSum(n, i_0)$
Полученное значение sum подставляется вместо параметра k, в правильный ответ записывается значение i_0. Параметры n, i_0 генерируются из условия $n - i_0 > 0$.

4. На какой итерации i значение S станет равным k?
$n = Gen(a, b)$
$i = Gen(a, b)$
$sum = FuncSum(i, 0)$
Полученное значение sum подставляется вместо параметра k, в правильный ответ записывается значение i. Параметры n, i генерируются из условия $n - i > 0$.

5. Сколько итераций было выполнено, если по завершении цикла значение $S = k$?
$n = Gen(a, b)$
$i_0 = Gen(a, b)$
$sum = FuncSum(n, i_0)$
Полученное значение sum подставляется вместо параметра k, в правильный ответ записывается значение $n - i_0$. Параметры n, i_0 генерируются из условия $n - i_0 > 0$.

Метод генерации меню вопросов

В тех случаях, когда необходимо проверить знание понятийного аппарата, определений терминов, можно использовать словарь. Например, пусть задан словарь, представленный следующей таблицей:

Байт	Ячейка памяти размером 8 бит
Слово	Ячейка памяти, которой оперирует аппаратная часть вычислительной машины
Регистр	Ячейка памяти процессора
Адрес	Номер ячейки памяти
Блок	Фрагмент памяти, превышающий размер слова

Можно получить следующие вопросы.

Вопрос первого типа:

Байт — это ...
1) ячейка памяти процессора
2) фрагмент памяти, превышающий размер слова
3) ячейка памяти, которой оперирует аппаратная часть вычислительной машины
4) ячейка памяти размером 8 бит

Укажите верный вариант ответа:

(Правильный ответ: 4.)

Вопрос второго типа:

Фрагмент памяти, превышающий размер слова, это — ...
1) байт
2) слово
3) блок
4) регистр

Укажите верный вариант ответа:

(Правильный ответ: 3.)

Вопрос третьего типа:

Ячейка памяти, которой оперирует аппаратная часть вычислительной машины, это — ... (введите термин).

(Правильный ответ: слово.)

Генератор, основанный на такой таблице, работает следующим образом:

1. Случайно выбирается i-я строка таблицы.
2. Определяется тип вопроса.
3. Если вопрос первого типа, то термин записывается в текст вопроса, далее в качестве вариантов ответа (пунктов меню) записываются правильное определение и три выбранных случайным образом определения из таблицы (эти определения будут неправильными).
4. Если вопрос второго типа, то определение записывается в текст вопроса, далее в качестве вариантов ответа (пунктов меню) записываются правильный термин и три выбранных случайным образом термина из таблицы (эти термины будут неправильными).
5. Для третьего типа записывается определение и требуется ввести термин.

Для **подсчета количества вариантов** для первого типа вопросов можно предложить следующую формулу:

$$v_1 = nC_{n-1}^{k-1},$$

где v_1 — количество вопросов первого типа, n — количество статей в словаре, k — количество вариантов ответов (пунктов меню).

Например, если $n = 5$, $k = 4$, общее число вариантов будет:

$$v_1 = 5\,\frac{4!}{1!\,3!} = 20.$$

Для второго типа вопросов формула будет аналогичной:

$$v_2 = nC_{n-1}^{k-1}.$$

Для третьего типа вопросов можно сгенерировать только n вопросов. Тогда общее число вопросов, которое можно сгенерировать предложенным алгоритмом, будет следующее:

$$v = v_1 + v_2 + v_3 = 2nC_{n-1}^{k-1} + n.$$

Для нашего примера общее число вопросов равно $20 + 20 + 5 = 45$. Очевидно, что при увеличении размера словаря количество вопросов будет существенно больше.

Генерация текста тестового задания

Дерево И-ИЛИ можно использовать для *многовариантного представления тестового задания* [3].

Для этого текст задания разбивается на фрагменты, фрагменты обычно разбиваются на классы — постоянные $\{T\}$ и переменные $\{V\}$. Для переменных фрагментов записываются множества реализаций, каждая из которых представляет собой конкретный текст. Затем каждый из выделенных фрагментов реализаций анализируется и, если есть возможность, разбивается на переменные и постоянные подфрагменты.

Рассмотрим использование данного метода на конкретном **примере**.

Пусть дано конкретное тестовое задание:

Дана следующая функция, написанная на языке программирования Си:

```
int FuncSum(int v[], int n){
  int sum=0;
  for(int i=0; i<n; i++){
    sum+=*v++; }
  return sum;}
```

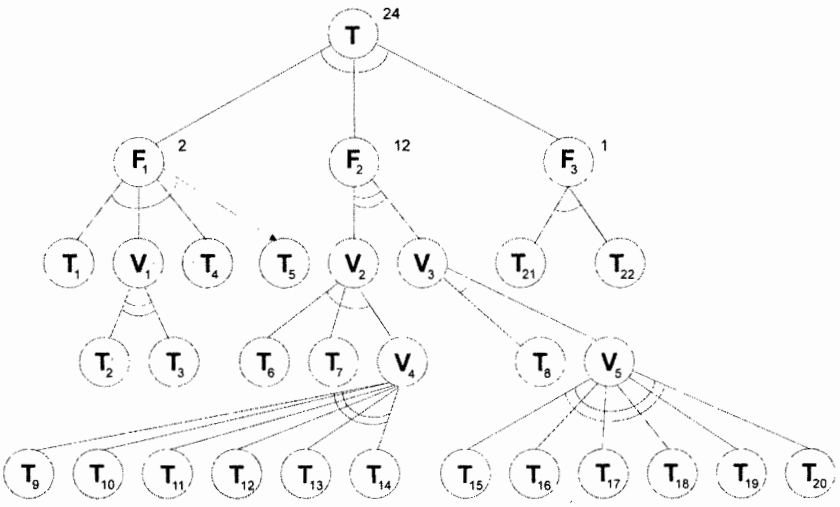
и следующий фрагмент программы:

```
int vec[5]={1, 7, 12, 5, 7};
printf("%s", FuncSum(vec, 5));
```

Какое число будет напечатано?

В этом задании учащемуся необходимо понять, что функция *FuncSum* определяет сумму элементов одномерного массива *v*, и в ответ ввести конкретный результат.

Используя выразительные возможности языка программирования Си, можно различными способами записать функцию нахождения суммы. Например: название функции может быть различным (*FuncSum*, *Total*, *CountSum*, *count_sum* и т. д.); описание параметров также может иметь различные вариации (*int *v*, *int v[]*); цикл можно записать с помощью оператора *while* или *for*; тело цикла также может быть записано различными способами. Таким образом, решение задачи нахождения суммы элементов одномерного массива можно реализовать некоторым множеством функций. Это множество можно представить деревом И-ИЛИ, показанным на рисунке.



Дерево И-ИЛИ для представления множества подпрограмм

Весь текст функции можно разделить на три фрагмента $\{F_1, F_2, F_3\}$, тогда узел T будет И-узлом, содержащим «сыновей» F_1, F_2, F_3. Каждый из фрагментов разделяется на фиксированные и переменные части. Например, узел F_1 разбивается на три фиксированных фрагмента $\{T_1, T_4, T_5\}$ и один переменный V_1, который имеет два варианта реализации $\{T_2, T_3\}$. Ниже перечислены значения узлов $\{T_i\}_{i=1}^{22}$:

```
<T1> := int FuncSum(int
<T2> := *v
<T3> := v[]
<T4> := ,int n){
<T5> := int sum=0;
<T6> := int i=0;
<T7> := while(i<n) {
<T8> := for(int i=0; i<n; i++){
<T9> := sum=sum+v[i++];
<T10>:= sum=sum+v[i]; i++;
<T11>:= sum=sum+*v++; i++;
<T12> := sum+=v[i]; i++;
<T13> := sum+=*v++; i++;
<T14> := sum=sum+*(v+i); i++;
<T15> := sum=sum+v[i];
<T16> := sum=sum+v[i];
<T17> := sum=sum+*v++;
<T18> := sum+=v[i];
<T19> := sum+=*v++;
<T20> := sum=sum+*(v+i);
<T21> := }
<T22> := return sum; }
```

Используя алгоритм подсчета вариантов в дереве И-ИЛИ [3], можно подсчитать общее количество вариантов, которое будет равно 24. Для генерации конкретной функции необходимо получить номер варианта и, используя алгоритм построения варианта [3], получить соответствующий вариант. Производя левосторонний обход этого варианта и записывая «листья», получим конкретное описание функции. Например:

Вариант {T1, T2, T4, T5, T6, T7, T9, T21, T22}:

```
int FuncSum(int *v,int n){
  int sum=0;
  int i=0;
  while(i<n) {
    sum=sum+v[i++];
    }
  return sum;
}
```

Вариант {T1, T3, T4, T5, T8, T19, T21, T22}:

```
int FuncSum(int v[],int n){
  int sum=0;
  for(int i=0; i<n; i++){
    sum+=*v++;
  }
  return sum;
}
```

Предложенные модели и алгоритмы для генерации вопросов и тестовых заданий могут быть использованы в различных компьютерных учебных программах. Можно предложить конкретные направления:

- тренажер для подготовки к единому государственному экзамену по информатике;
- тестовая программа по курсу информатики для организации самоконтроля;
- генератор индивидуальных заданий по курсу информатики;
- тестовые контрольные работы по различным разделам курса информатики.

Данные модели и алгоритмы опробованы при создании компьютерных контрольных работ и экзаменов по информатике в Томском межвузовском центре дистанционного образования [3].

Литература

1. *Кручинин В. В.* Разработка компьютерных учебных программ. Томск: Изд-во Томск. ун-та, 1998.

2. *Башмаков А. И., Башмаков И. А.* Разработка компьютерных учебников и обучающих систем. М.: Информационно-издательский дом «Филинъ», 2003.

3. *Кручинин В. В.* Генераторы в компьютерных учебных программах. Томск: Изд-во Томск. ун-та, 2003.

4. *Исакова О. Ю., Кручинин В. В.* Методика получения вопросов для компьютерного тестирования // Компьютерные инструменты в образовании. 2004. № 3.

5. *Кручинин В. В., Морозова Ю. В.* Модели и алгоритмы генерации задач в компьютерном тестировании // Известия Томского политехнического университета. 2004. № 5.

6. *Кручинин В. В., Морозова Ю. В.* Модели генераторов вопросов для компьютерного контроля знаний // Открытое и дистанционное образование. 2004. Вып. 2(14).