Компьютерные инструменты в образовании, 2024
№ 1: 71–84
УДК: 004.42
http://cte.eltech.ru
doi:10.32603/2071-2340-2024-1-100

МЕТОД ГЕНЕРАЦИИ УНИКАЛЬНЫХ ВАРИАНТОВ ДЛЯ МАТЕМАТИЧЕСКИХ ЗАДАЧ

Винокурова Д. В.[1], аспирант, ✉ d.v.vinokurova@gmail.com

[1]Российский государственный педагогический университет им. А. И. Герцена, набережная реки Мойки, д. 48, 191186, Санкт-Петербург, Россия

Аннотация

В рамках статьи ставится проблема создания генераторов для уникальных многовариантных математических задач. Рассматриваются методы получения случайных чисел. Проводится сравнительный анализ стандартного генератора, применяемого разработчиками, и генератора, предоставляющего возможность создания наборов чисел, встречающихся один раз среди требуемого диапазона значений. Предлагается конкретный пример, для которого требуется применение уникальных наборов задач и параметров. Пример сопровождается формулами в общем виде, листингами предлагаемых алгоритмов функций на языке JavaScript, кратким словесным описанием результата работы функций. Уделяется внимание генерации чисел без ноля, применяющейся для генерации параметров, которые не нужно обращать в ноль. Рассматриваются особенности разработки с учетом выбора наилучшего интервала для широкого диапазона значений генерируемых чисел, исключая применение циклов, увеличивая скорость выполнения программы. Объясняется способ описания результата генерации чисел в формате JSON, который можно использовать при передаче полученных наборов между различными языками программирования.

Ключевые слова: *генерация задач, псевдослучайные числа, уникальные варианты, многовариантные задачи, типовые задачи.*

Цитирование: Винокурова Д. В. Метод генерации уникальных вариантов для математических задач // Компьютерные инструменты в образовании. 2024. № 1. С. 71–84. doi:10.32603/2071-2340-2024-1-100

1. ВВЕДЕНИЕ

Типовые многовариантные задачи по предметам математического цикла являются неотъемлемой частью процесса обучения. Важным фактором выступает особенность их разработки, требуется, чтобы они являлись уникальными. В генераторах обычно присутствуют математические выражения с параметрами или функциями, которые можно изменять. Генераторы входят в основу многих известных языков программирования, программных разработок [1, 2], алгоритмов генерации с использованием среды Mathematica [3, 4]. Использование стандартных генераторов, входящих в состав современных языков программирования, не исключает генерацию повторяющихся значений, и при генерации типовых заданий нужно учитывать последовательности с неповторяющимися элементами.

В статье предлагается метод генерации параметров для создания уникальных наборов для различных математических задач. Примеры сопровождаются кодом на языке JavaScript.

Целью настоящей работы является сравнение особенностей генератора случайных чисел, часто используемого разработчиками, с особенностями предлагаемого генератора, а также описание возможностей представленного метода генерации уникальных вариантов с различным набором параметров.

2. МЕТОД ГЕНЕРАЦИИ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ БЕЗ ПОВТОРЕНИЙ

По способу получения случайных значений генераторы случайных чисел (ГСЧ) делятся на физические, табличные и алгоритмические. Физические ГСЧ получаются в результате применения устройства, генерирующего случайные числа на основе изменения физического процесса. Табличные ГСЧ составляются из специальных таблиц, в которых содержатся некоррелированные числа. Данную таблицу можно обходить всеми возможными способами. Алгоритмические ГСЧ используют некий алгоритм для генерации чисел [5].

Генерации задач посвящены многие диссертационные исследования. В. В. Кручинин уникальность создаваемых вариантов обеспечивает за счет генерации задач на основе деревьев И/ИЛИ [6]. А. В. Титков, опираясь на исследования Кручинина вводит теоретико-множественные операции на деревьях И/ИЛИ и предлагает в качестве способа создания генераторов язык GIL [7]. Ю. А. Зорин расширяет метод представления тестовых заданий, предложенных В. В. Кручининым, вводя новые узлы и понятия, и предлагает язык GILT для создания алгоритмов генерации [8]. И. А. Посов в своем исследовании [9] предоставляет метод создания генераторов при помощи предметно-ориентированного языка, который автор называет Possum. Данный язык содержит генераторы случайных чисел и различных комбинаторных объектов [9, с. 43] среди которых присутствует случайное дерево И/ИЛИ. Автор не заостряет внимание на уникальности создаваемых задач, не приводит алгоритмов. Неповторяемость задач, по-видимому, обеспечивается преподавателями, разрабатывающими генераторы при помощи Possum.

Создание генератора задач подразумевает наличие генератора псевдослучайных чисел (ГПСЧ), основанных на специальном алгоритме, который должен формировать целые числа. В языках программирования существуют генераторы, которые осуществляют генерацию чисел на интервале $[a, b)$, где $a = 0$, $b = 1$, числа, входящие в этот интервал, являются рациональными. Для получения целочисленных значений многие разработчики, в том числе, автор книги Майл МакГрат [10] практикуют прием, который позволяет получать данные числа путем округления методами round( ), floor( ) или ceil( ), которые встречаются во всех языках программирования.

Рассмотрим пример кода для генерации случайных чисел (см. Algorithm 1). Для удобства путем дополнительных арифметических операций увеличиваем диапазон генерации и осуществляем возможность получения целых случайных чисел на отрезке $[a, b]$, где a — минимальное значение, b — максимальное.

С целью демонстрации работы генератора проведем эксперимент. Будем генерировать различные числа: от 1 до m, где m принимает значения из диапазона $[10, 20, 30]$. Сгенерированные значения будем хранить в массиве.

Algorithm 1:

```
// * Генерация целочисленных чисел от min до max значений
// ******************************************************
function rndInt(min, max){
   let rand = Math.random()*(max - min) + min;
   return Math.round(rand);
}
```

Количество сгенерированных чисел без повторений рассчитывается по комбинаторной формуле размещений без повторений (1), в данном случае параметр n обозначает количество генерируемых чисел, параметр k равен 1, так как сгенерированное число может встречаться только один раз.

$$A_n^k = \frac{n!}{(n-k)!}. \tag{1}$$

Перепишем формулу (1) для рассматриваемого случая в виде (2). В результате получим, что для n различных чисел, встречающихся ровно один раз, можно получить n чисел. Применяя формулу (2) для значений из диапазона $[10, 20, 30]$ получим значения 10, 20, 30 соответственно.

$$A_n^1 = \frac{n!}{(n-k)!} = \frac{1 \cdot \ldots \cdot (n-1) \cdot n}{1 \cdot \ldots \cdot (n-1)} = n. \tag{2}$$

Используя библиотеку Chart.js, предназначенную для построения графиков и диаграмм с помощью языка JavaScript, получим гистограммы (см. рис. 1–2), где по оси абсцисс отражены генерируемые числа от 1 до n, по оси ординат — количество чисел, встречающихся в выборке.

Мы видим, что числа неравномерно распределены по отрезку и возникли повторы. Первая гистограмма (см. рис. 1) содержит 10 чисел, из которых число 4 встречается три раза, числа 2 и 9 — два раза, числа 1, 3, 6, 7 не попали в выборку. На второй гистограмме (см. рис. 2 а) преобладает количество чисел 4 и 20, числа 17 и 18 встречаются по два раза, числа 1, 6, 10, 14, 15 не попали в выборку. Последняя гистограмма (см. рис. 2 б) особенно четко показывает разброс и неравномерность распределения, таким образом, настоящий генератор не создаст действительно уникальный набор вариантов.

Рис. 1. Эксперимент с выборками для стандартного генератора

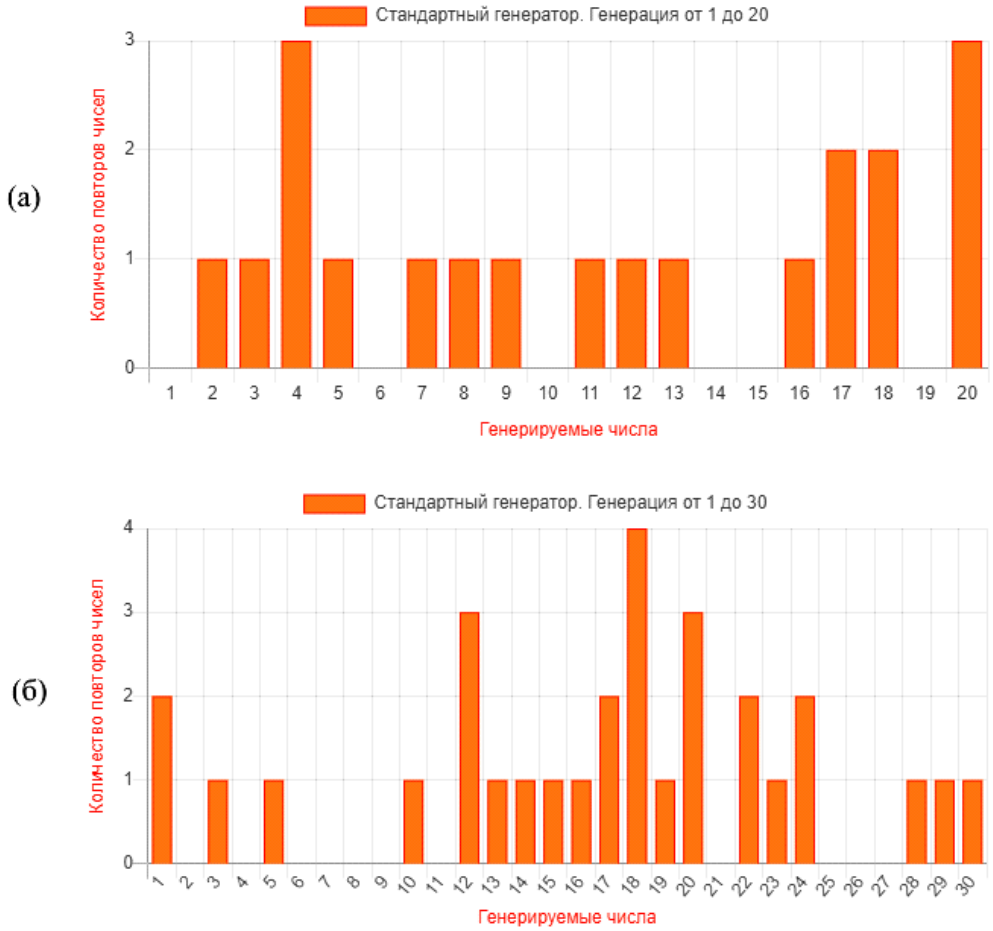

Рис. 2. Эксперимент с выборками для стандартного генератора

В целях предотвращения данной ситуации предлагается использовать следующий алгоритм (см. Algorithm 2). Опираясь на стандартный алгоритм (функцию rndInt в Algorithm 1) выполняются некоторые дополнительные операции. Создаются два массива: массив чисел arrRndValues и массив уникальных чисел arrUniformNum. В arrUniformNum в цикле добавляются только числа, которых нет в данном массиве. За счет этого мы добиваемся уникальности. В arrRndValues добавляются все числа из диапазона от minNum до maxNum. Используемый массив является условием выхода из цикла, как только будет достигнут требуемый диапазон генерируемых чисел.

Функция rndNZ в условных операторах genUniformIntNumbers вызывается, если в процессе генерации число 0 необходимо исключить. В этой функции происходит вызов алгоритма обработки интервалов, который выбирает интервал с большим количеством чисел слева или справа от нуля, что увеличивает скорость генерации уникальных чисел. Возможности работы данного алгоритма будут описаны ниже.

Повторно проведенный эксперимент с выборками иллюстрируют гистограммы на рис. 3–5. Гистограммы для генерации чисел при помощи предлагаемого алгоритма (см. рис 3–5) и с использованием стандартного генератора значительно различаются.

Algorithm 2:

```
// * genUniformIntNumbers - генератор уникальных целых чисел
// * minNum  - минимальное число
// * maxNum  - максимальное число
// * quantityNumbers - количество чисел
// * zero - включать 0 или нет
// ********************************************
function genUniformIntNumbers(minNum, maxNum, quantityNumbers, zero) {
   let arrUniformNum;
   let arrRndValues;
   let rndNum;

   arrUniformNum = [];
   arrRndValues = [];
   do {
      // проверка включения нуля
      if (!zero) {
         rndNum = rndNZ(minNum, maxNum);
      } else {
         rndNum = rndInt(minNum, maxNum);
      }

      if (arrUniformNum.length == quantityNumbers) {
         arrUniformNum = [];
      }
      if (!arrUniformNum.includes(rndNum)) {
         if (rndNum === -0) {
            rndNum = 0;
         }
         arrRndValues.push(rndNum);
         arrUniformNum.push(rndNum);
      }
   } while (arrRndValues.length != quantityNumbers);
   return arrRndValues;
}
```

Функция genUniformIntNumbers обеспечила генерацию уникальных выборок для 10, 20, 30 значений, в которых каждое число встречается ровно один раз. Отметим, что для создания многовариантных задач требуются дополнительные функции (на гистограммах представлены эксперименты только для одной выборки для каждого из значений).

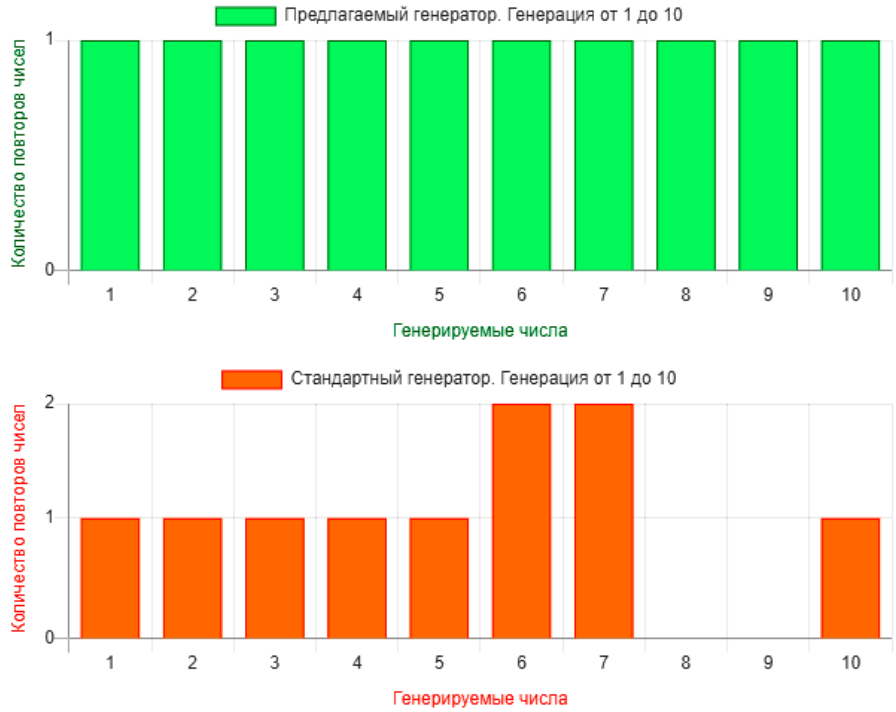

Рис. 3. Эксперимент с выборкой для сравнения предлагаемого генератора со стандартным на примере 10 чисел

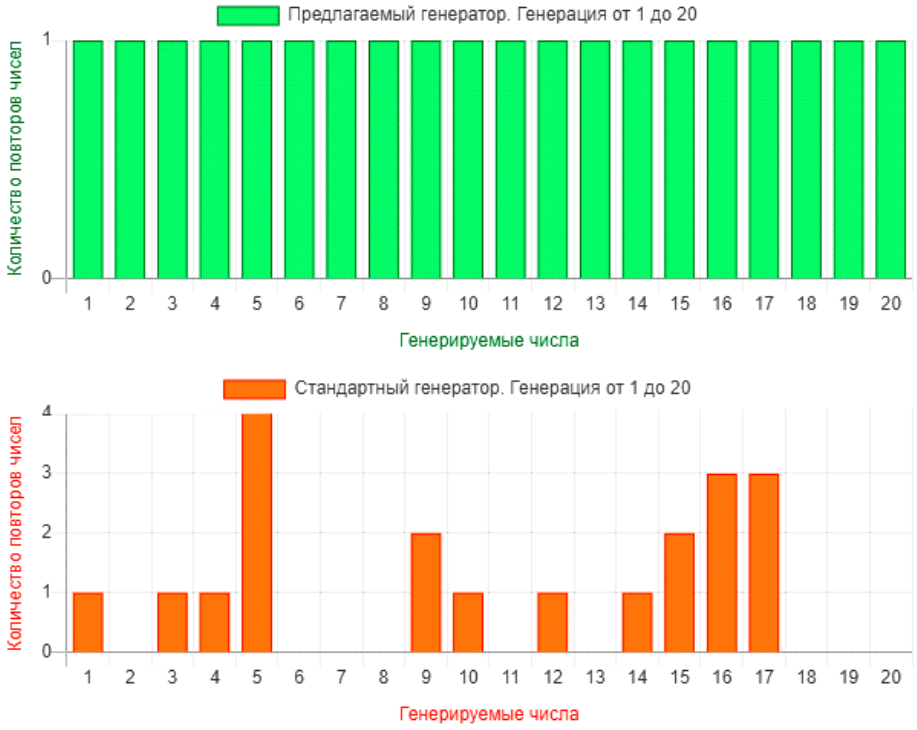

Рис. 4. Эксперимент с выборкой для сравнения предлагаемого генератора со стандартным на примере 20 чисел

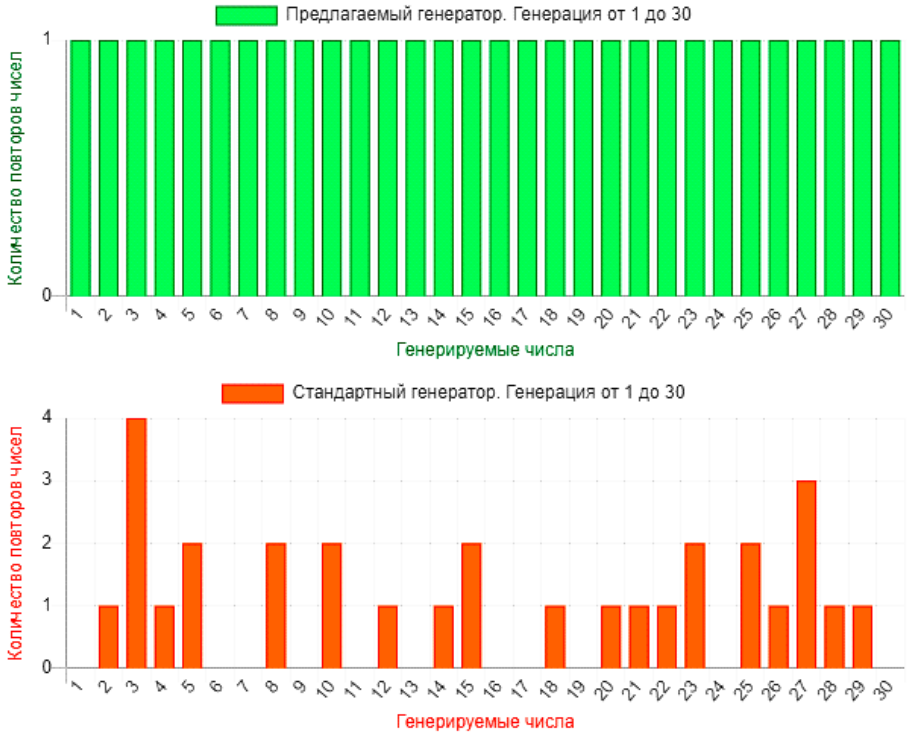

Рис. 5. Эксперимент с выборкой для сравнения предлагаемого генератора со стандартным на примере 30 чисел

3. ГЕНЕРАЦИЯ УНИКАЛЬНЫХ НАБОРОВ С ПАРАМЕТРАМИ

Для лучшего понимания процесса генерации приведем конкретный пример. Допустим, для генерации заданий некоторого типа необходимо получить 7 наборов (вариантов) заданий, в которых используется три ненулевых параметра a, b, c и требуется, чтобы выборки не повторялись. Выполнение данных требований требует использование функции, которая позволяет задавать диапазон чисел, параметры, количество наборов для генерации. Основные части алгоритма данной функции, именуемой setsNumbers, проиллюстрированы в Algorithm 3.

Массив sets в общем виде представлен формулой (3) и для рассматриваемого примера состоит из 7 массивов (наборов), в которых содержатся уникальные числа для параметров prm_{kt} из t значений для k наборов. Уникальность чисел обеспечивается при помощи применяемого выше алгоритма для функции genUniformIntNumbers.

$$\mathrm{sets}_1 = \underbrace{\Big[\big[\mathrm{prm}_{11}, \ldots, \mathrm{prm}_{1t}\big], \ \ldots, \ \big[\mathrm{prm}_{k1}, \ldots, \mathrm{prm}_{kt}\big]\Big]}_{k \text{ наборов (вариантов) для каждого из } m \text{ параметров}}. \tag{3}$$

Исключительность наборов обеспечивается предварительным расчетом количества вариантов для заданных значений и параметров (функция placementWithoutDuplicate с соответствующими параметрами n и k) при помощи формулы размещений без повторений (1). Рассмотрение алгоритма этой вспомогательной функции для вычислений из-за простоты реализации опустим. В формуле (1) для данной задачи параметр n обозначает количество генерируемых чисел с нулем или без нуля, k — количество параметров.

Algorithm 3: Создание уникальных наборов ПСЧ для параметров

```
// Функция позволяет создавать уникальные наборы ПСЧ для параметров
// ---------------------------------------------------------------
// * min - минимальное число
// * max - максимальное число
// * prm – параметры в виде [“a”,”b”,”c”, ...]
// * quantitySets - количество необходимых наборов (вариантов)
// * zero значения true/false, использовать 0 или нет
// *******************************************************
function setsNumbers(min, max, prm, quantitySets, zero) {
   let quantityNum;
   et quantityPrm;
   let sets;
   let currentSet;
   let coeffJSON;
   let lenPrm;

   quantityPrm = prm.length;

   // * подсчет чисел в зависимости от того, включается 0 в интервал или нет
   quantityNum = quantityNumInInterval(min, max, zero);

   if (quantityPrm == 0) {
      console.log("Ошибка! Необходимо указать параметры!");
      return [];
   }
   else if(quantityNum < quantityPrm)
   {
      console.log("Ошибка! Число параметров больше числа генерируемых чисел!");
      return [];
   }
   else if(quantitySets <= placementWithoutDuplicate(quantityNum, quantityPrm))
   {
      let setsWords;
      let currentElem;
      let currentElemWord;
      sets = [];
      setsWords = [];
      // * значения для параметров исключая повторы
      let j = 0;
      while (j < quantitySets) {
         for (let i = 0; i < quantityPrm; i++) {
            currentElem = genUniformIntNumbers(min, max, quantityPrm, zero);
            currentElemWord = currentElem
            .map((object) => String(object))
            .join("");
            // проверка включения сгенерированной последовательности
            // среди остальных
            if (!setsWords.includes(currentElemWord)) {
               sets.push(currentElem);
               // преобразование элемента массива в строку
```

```
                setsWords.push(currentElemWord);
                j++;
            }

            if (j == quantitySets) {
               break;
            }
         }
      }

      // распределение наборов с параметрами по готовым наборам
      coeffJSON = {};
      for (let j = 0; j < quantitySets; j++) {
         currentSet = {};
         for (leti = 0; i < quantityPrm; i++) {
            currentSet[prm[i]] = sets[j][i];
         }
         coeffJSON[j + 1] = currentSet;
         }

      coeffJSON = JSON.stringify(coeffJSON, null, 2);// JSON формат

      return coeffJSON;
   }
}
```

В качестве примера рассмотрим вычисление (4) необходимого количества уникальных наборов для исходных данных (см. Algorithm 4) по формуле (1) для пяти чисел и трех параметров. Видно, что с такими исходными данными можно сгенерировать 60 уникальных вариантов.

$$A_5^3 = \frac{5!}{(5-3)!} = \frac{5!}{2!} = 3 \cdot 4 \cdot 5 = 60. \tag{4}$$

Algorithm 4:

```
minNum = 1;
maxNum = 5;
prm = ["a", "b", "c"];
```

В функции (см. Algorithm 3) в переменной currentElem генерируется уникальный массив с параметрами, в переменной currentElemWord хранится массив, преобразованный в строку, для упрощения дальнейшей проверки наличия данного набора в массиве. Массив setsWords содержит все сгенерированные наборы в виде строк. В условии проверяется существование набора currentElemWord в массиве setsWords. В случае отсутствия этот набор добавляется в массивы setsWords и sets.

В качестве возвращаемого результата этой функции выступает объект coeffJSON, который хранит сгенерированные параметры в JSON формате записи (5), что позволяет их

дальнейшее использование в других языках программирования. В формуле (5) номер варианта (набора) обозначается s_k, prm_{kt} — текущий параметр, value_prm_{kt} — значение параметра.

$$\text{coeffJSON} = \begin{array}{l} \{ \\ s_1 : \{\text{prm}_{11} : \text{value\_prm}_{11},\ ..., \text{prm}_{1t} : \text{value\_prm}_{1t}\}\} \\ ..., \\ s_k : \{\text{prm}_{k1} : \text{value\_prm}_{k1},\ ..., \text{prm}_{kt} : \text{value\_prm}_{kt}\}\} \\ \} \end{array} \tag{5}$$

В продолжение примера генерации 7 наборов для трех ненулевых параметров рассмотрим скриншот (см. рис. 6) выполнения функции setsNumbers при запуске функции в виде: setsNumbers(1, 5, ["a", "b", "c"], 7, false).

generator-int-num.js:357


```
{
  "1": {
    "a": 1,
    "b": 2,
    "c": 3
  },
  "2": {
    "a": 2,
    "b": 5,
    "c": 3
  },
  "3": {
    "a": 2,
    "b": 1,
    "c": 4
  },
  "4": {
    "a": 3,
    "b": 2,
    "c": 4
  },
  "5": {
    "a": 5,
    "b": 3,
    "c": 2
  },
  "6": {
    "a": 3,
    "b": 5,
    "c": 2
  },
  "7": {
    "a": 2,
    "b": 5,
    "c": 4
  }
}
```


Рис. 6. Результат работы функции setsNumbers

Из приведенного примера видно, что параметры в форме JSON не повторяются и уникальны для всех 7 вариантов.

4. АЛГОРИТМ ОБРАБОТКИ ИНТЕРВАЛОВ

Параметры не всегда обладают положительными значениями, они могут принимать также и отрицательные значения, что вызывает необходимость учитывать возможность включения или исключения значения нуля в интервал.

Отсутствие нуля в интервале делит его на две части $[n_1, 0)$ и $(0, n_2]$. В каждом из этих интервалов может быть как большее, так и меньшее количество чисел. Например, $[-2, 9] = [-2, -1] \cup [1, 9]$ на отрезке $[-2, -1]$ содержится два значения, на отрезке $[1, 9]$ включено девять значений. Для приближения к минимуму повторов чисел в получаемой выборке и увеличения скорости работы программы целесообразно генерировать значения из отрезка с большим количеством чисел.

Вызов функции rndNZ (см. Algorithm 5) внутри функции genUniformIntNumbers (см. Algorithm 2) происходит в случае, когда необходимо, чтобы параметр не обращался в ноль. Функция rndNZ вызывает известную функцию rndInt и в случае генерации нуля вызывает функцию selectInterval, обеспечивая выбор нужного интервала.

Algorithm 5:

```
// * Генерация чисел без 0
function rndNZ(min, max) {
   let num = rndInt(min, max);
   return num === 0 ? selectInterval(min, max) : num;
}
```

Необходимость использования selectInterval обосновывается тем, что в случае получения нуля данная функция вернет число из интервала с большим количеством чисел. Если не применять данную функцию, то для получения ненулевого числа могут попадаться значения из отрезка с меньшим количеством чисел, что будет приводить к появлению лишних итераций в цикле в функции genUniformIntNumbers, поскольку необходим уникальный набор, а не повторяющиеся числа. Например, в отрезке $[-2, 9]$ могут несколько раз попадаться значения $-2, 0, -1$. В случае если -2 и -1 уже есть в выборке, необходимо будет повторно генерировать значения, что приведет к увеличению времени выполнения программы, не говоря о выпадении числа 0, которое в данном случае требуется исключить.

Для лучшего понимания код данной функции приводить не будем, а кратко опишем суть ее работы. Функция selectInterval для различных интервалов позволяет выбрать тот, который наилучшим образом подходит для каждой конкретной ситуации. Частными случаями являются следующие интервалы: $[-1, n_2]$ и $[n_1, 1]$. Опишем эти интервалы подробно и представим результирующий интервал, который будет подан на вход другой функции для генерации значений.

$$1.[-1, n_2] = \underbrace{[-1, 0)}_{1\text{ число}} \cup \underbrace{(0, n_2]}_{n_2 - 1\text{ число}} \xrightarrow{\text{интервал для генерации}} [1, n_2],$$

$$2.[n_1, 1] = \underbrace{[n_1, 0)}_{n_1 - 1\text{ число}} \cup \underbrace{(0, 1]}_{1\text{ число}} \xrightarrow{\text{интервал для генерации}} [n_1, -1].$$

В случае если интервал представлен в виде $[n_1, n_2]$, необходимо определить принадлежность n_1 и n_2 к положительному или отрицательному интервалам. При условии n_1

меньше нуля и n_2 больше нуля выбирается интервал с большим количеством чисел, как в примерах ниже:

$$[-9,2] = \underbrace{[-9,0)}_{\text{9 чисел}} \cup \underbrace{(0,2]}_{\text{2 числа}} \xrightarrow{\text{интервал для генерации}} [-9,-1],$$

$$[-2,9] = \underbrace{[-2,0)}_{\text{2 числа}} \cup \underbrace{(0,9]}_{\text{9 чисел}} \xrightarrow{\text{интервал для генерации}} [1,9].$$

При одинаковом количестве значений в обоих интервалах выбирается случайный. Для случаев, когда оба числа больше нуля, интервалы для генерации остаются исходными. В случае если один из интервалов содержит 0: $[0, n_2]$ или $[n_1, 0]$, к 0 прибавляется или отнимается значение 1, и интервалы принимают вид $[1, n_2]$ или $[n_1, -1]$.

Применение данной функции отбора интервалов позволяет увеличить скорость работы программы за счет устранения лишних итераций и служит надежным способом описания требуемых действий при таком подходе, поскольку в условных операторах описываются все возможные случаи, которые могут произойти в интервалах.

5. ЗАКЛЮЧЕНИЕ

Обзор литературы показал, что авторы для создания уникальных наборов задач прибегают к использованию деревьев И / ИЛИ. Исходя из сравнительного анализа гистограмм генераторов, представленных в данной статье, можно сделать вывод, что стандартный генератор не учитывает особенности разработки генераторов с уникальным набором значений. Использование алгоритма предложенного генератора позволяет сформировать необходимые условия для создания собственных генераторов математических задач. Вычисление количества выборок с использованием комбинаторики исключит возможность повторяющихся наборов. Применяемая обработка интервалов обеспечит увеличение скорости генерации значений и не допустит возникновение ошибок. Предоставление сгенерированного результата в JSON формате позволит разработчикам использовать данный подход при реализации генераторов на различных языках программирования. Метод генерации многовариантных заданий обеспечит преподавателям возможность разработки качественных уникальных наборов заданий.

Список литературы

1. *Финогенов А. А.* Использование генератора задач для контроля знаний по высшей математике у студентов младших курсов // Вестник ЮГУ. 2016. № 2 (41). С. 65–67.
2. *Винокурова Д. В.* Информационная система для обучения решению квадратных уравнений с параметром // Международная научно-практическая интернет-конференция «Актуальные проблемы методики обучения информатике и математике в современной школе». М., 2023. URL: http://news.scienceland.ru/2023/04/23/информационная-система-для-обучения/ (дата обращения 01.12.23).
3. *Сосновский Н. Н.* Разработка методических материалов в среде системы Mathematica // Компьютерные инструменты в образовании. 2015. № 5. С. 53–60.
4. *Иванова Н. А.* Возможные направления применения ресурсов программирования среды Mathematica при решении математических задач // Вестник Балтийского федерального университета им. И. Канта. Серия: Филология, педагогика, психология. 2012. № 5. С. 155–160.
5. *Мухин О. И.* Моделирование систем. Пермь: ПГТУ, 2001. URL: stratum.pstu.ac.ru (дата обращения 01.12.23).

6. *Кручинин В. В.* Методы, алгоритмы и программное обеспечение комбинаторной генерации // Докт. диссертация, 2010. 280 с.
7. *Титков А. В.* Система построения генераторов комбинаторных множеств на основе деревьев И/ИЛИ // Канд. диссертация, 2010. 150 с.
8. *Зорин Ю. А.* Автоматизация построения многовариантных тестовых заданий на основе деревьев И/ИЛИ // Канд. диссертация, 2014. 139 с.
9. *Посов И. А.* Автоматизация процесса разработки и использования многовариантных учебных заданий / Канд. диссертация, 2012. 134 с.
10. *М. МакГрат.* JavaScript для начинающих. М: Эксмо, 2023. 232 с.

Поступила в редакцию 03.12.2023, окончательный вариант — 10.02.2024.

Винокурова Дарья Валентиновна, аспирант, Институт информационных технологий и технологического образования, РГПУ, ✉ d.v.vinokurova@gmail.com

Computer tools in education, 2024
№ 1: 71–84
http://cte.eltech.ru
doi:10.32603/2071-2340-2024-1-100

Method of Generating Unique Variants for Mathematical Problems

Vinokurova D. V.[1], Postgraduate, ✉ d.v.vinokurova@gmail.com

[1]Herzen University, 48 Moika river embankment, 191186, Saint Petersburg, Russia

Abstract

The article poses the problem of creating generators for unique multivariate mathematical problems. Types of methods of obtaining random numbers are considered. A comparative analysis of the standard generator used by developers with the generator providing the possibility to create sets of numbers occurring once among the required range of values is carried out. A specific example is provided that requires the application of unique problem sets and parameters. The examples are accompanied by formulas in general form, listings of the proposed algorithms of functions in JavaScript, a brief verbal description of the result of the functions. Attention is paid to the generation of non-zero numbers, used to generate parameters that do not need to be converted to zero. The features of development are considered, taking into account the choice of the best interval for a wide range of values of generated numbers, eliminating the use of loops, increasing the speed of program execution. The article explains a method of description of the result of generation of numbers in JSON format, which can be used when transferring the obtained sets between different programming languages.

Keywords: *problem generation, pseudo-random numbers, unique variants, multivariate problems, typical problems.*

Citation: D. V. Vinokurova, “Method of Generating Unique Variants for Mathematical Problems,” *Computer tools in education*, no. 1, pp. 71–84, 2024 (in Russian); doi:10.32603/2071-2340-2024-1-100

References

1. A. A. Finogenov, “Ispol’zovanie generatora zadach dlya kontrolya znanij po vysshej matematike u studentov mladshih kursov” [Using a task generator to control the knowledge of higher mathematics among undergraduate students], *Vestnik YUGU*, no. 2 (41), pp. 65–67, 2016 (in Russian).
2. D. V. Vinokurova, “Informacionnaya sistema dlya obucheniya resheniyu kvadratnyh uravnenij s parametrom” [Information system for learning to solve quadraticequations with parameters], in *Mezhdunarodnaya nauchno-prakticheskaya internet-konferenciya Aktual’nye problemy metodiki obucheniya informatike i matematike v sovremennoj shkole», 24-28 Apr. 2023*, Moscow, 2023 (in Russian). [Online]. Available: http://news.scienceland.ru/2023/04/23/информационная-система-для-обучения/
3. N. N. Sosnovskij, “Razrabotka metodicheskih materialov v srede sistemy Mathematica” [Development of methodical materialsin the environment of "mathematica"], *Computer tools in education*, no. 5, pp. 53–60, 2015 (in Russian).
4. N. A. Ivanova, “Vozmozhnye napravleniya primeneniya resursov programmirovaniya sredy Mathematica pri reshenii matematicheskih zadach” [Possible areas of application of Mathematica programming resources in solving mathematical problems], *Vestnik Baltijskogo federal’nogo universiteta im. I. Kanta. Seriya: Filologiya, pedagogika, psihologiya*, no. 5, pp. 155–160, 2012 (in Russian).
5. O. I. Muhin, “Modelirovanie sistem” [System Modeling], in *https://stratum.ac.ru* 2001(in Russian). [Online]. Available: https://stratum.ac.ru/education/textbooks/modelir/
6. V. V. Kruchinin, “Metody, algoritmy i programmnoe obespechenie kombinatornoj generacii” [Methods, algorithms and software for combinatorial generation], *Doctor Sc. diss., TUSUR, Tomsk, Russia*, 2013 (in Russian).
7. A. V. Titkov, “Sistema postroeniya generatorov kombinatornyh mnozhestv na osnovederev’ev I/ILI” [The system of construction of combinatorial set generators based on AND/OR trees], *Cand. Sc. diss., TUSUR, Tomsk, Russia*, 2010 (in Russian).
8. Y. A. Zorin, “Avtomatizaciya postroeniya mnogovariantnyh testovyh zadanij na osnove derev’ev I/ILI” [Automation of multivariate test case construction based on AND/OR trees], *Cand. Sc. diss., TUSUR, Tomsk, Russia*, 2014 (in Russian).
9. I. A. Posov, “Avtomatizaciya processa razrabotki i ispol’zovaniya mnogovariantnyh uchebnyh zadanij” [Automating the development and use of multivariate training tasks], *Cand. Sc. diss., SPbSU, St. Petersburg, Russia*, 2012 (in Russian).
10. M. MakGrat, *JavaScript in easy steps*, Moscow: Eksmo, 2023 (in Russian).

Received 03-12-2023, the final version — 10-02-2024.

Daria Vinokurova, Postgraduate, Institute of Computer Science and Technology Education, ✉ **d.v.vinokurova@gmail.com**