Mathematical and Software Engineering, Vol. 1, No. 1 (2015), 12-17
Varεpsilon Ltd, http://varepsilon.com

Online Math Tasks Generator

**Julia Dimitrova, Eleonora Pavlova,
Kosyo Nikolov, and Aleksander Bonin**

High School of Mathematics "D-r Petar Beron", Varna, Bulgaria

Abstract

This article presents "Math for all" – a web application that generates various algebraic equations and inequalities. We discuss algorithms and methods that are implemented in building the library that generates all the expressions, and some problems that are solved in the process of our work.

Subject Codes (ACM): G.4, H.3.5.

Keywords: online generator, algebraic expressions, equations and inequalities, restrictions for answers and quotients.

1 Introduction

This article presents "Math for all" – a web application that generates various algebraic equations and inequalities. The generator can be used by Math teachers and everyone who wants to better their skills in this particular part of mathematics.

One of the main advantages of "Math for all" is the Bulgarian interface. The application is very easy to use and provides various settings through which the user can change the type of the equations and inequalities. The user can set the difficulty of the problem, the type of visualization of the answer, the number and the names of the variables, the power of the polynomials or roots, the overall complexity, and also they can choose the type of the answer. The program can generate a number of similar tasks. It solves every generated task and visualizes the result.

The application can be used online and is freely available. The intuitive user interface allows our application to be used even by people who are not experienced computer users. The most complex and interesting part of this application is the so-called "core" – a library which generates algebraic expressions, equations and inequalities. The description of the "core" is the main feature of this article.

2 Methods and Algorithms

The application program was created by two 10th grade students at High School of Mathematics who have in-depth knowledge of programming. KN is an active participant in C/C ++ programming competitions and AB is a contestant in various IT competitions.

The description of used data structures

The core of the application generates and calculates algebraic expressions. It is written in C ++. A hierarchy of classes is created for the implementation of the algorithms - classes ranging from simple monomials (e.g. x^2) to complex expressions and for specific tasks (Equation - equation, Inequation - inequality). STL (Standard Template Library) [5] -containers vector and map are used too.

Algorithms

Many of the algorithms used in calculations were created by the authors with the help of [1-4]. Optimizations were made to improve the speed of calculation. For example, the operation "multiplication of polynomials" is optimized into quick sorting, which shortens the computation time. Therefore, the maximum complexity is $O(N^2.logN)$.

Generating tasks is easier after a shaping algebraic basis. Appropriate methods are provided for this in class **Polynomial**. The "heavy lifting", so to say, is done by the class named **Polynomial**. It provides essential functions, such as addition, subtraction, multiplication and exponentiation of algebraic expressions. Computing (4x+3/2y)*(23y+x)-(x+3)^2 in a single line of code greatly decreases the difficulty of doing more advanced operations like the generation of math problems. **Polynomial** uses a lot of smaller classes to avoid the clustering of code that is too complex to understand. The hierarchy looks like this: Number -> Simple -> Monomial -> Polynomial.

The **Number** class is exactly what it sounds like - a number. It is the implementation of the coefficients we use in **Algebra**. Primitive number types cannot provide the functionality needed for the program, such as fractions. Floating point types would give an estimate, but **Number** keeps the fractions

intact, so 2/3 will not become 0.66667. In addition, **Number** can be upgraded to provide even more features, such as irrational coefficients (roots, logarithms and so on). This is not hard to do, as every other class uses **Number** by default (with some minor exceptions), so all the changes required would be within **Number** itself.

Simple is a class that denotes a letter with a power attached to it, for example x^2. Its most important feature is that it can be compared in a fashion similar to a letter.

Monomial is a class for manipulating monomials (such as 5/3x^2y^3). It contains a vector of Simples and a Number coefficient. Monomials can also be compared. This is done by a method you would normally use in a string, i.e. going from left to right and comparing each letter - in this case, every **Simple**. This feature of **Monomial** is crucial for the fast operation of Polynomial.

As you may have guessed, **Polynomial** contains a vector of **Monomials**. Those monomials are always sorted. Let's look into some of the operations we can perform:

Addition: we use a modified version of the algorithm for the merging of two sorted arrays. We start with 2 indexes, both at the first monomial of the two polynomials. Then we advance 1 position at a time: if the monomials at the respective indexes are the same, we add up their coefficients. If one is greater, we only advance the other index. We repeat this until we have used all the monomials. This gives us an *O(N)* performance. If the program were to use non-sorted vectors, it would have been $O(N^2)$. Subtraction is done the same way.

Multiplication uses the long multiplication algorithm, which has a complexity of $O(N^2)$ for numbers. There are algorithms with better performance; however, one should note that they only perform better in practice for a very large number of digits (in our case, monomials). No one would need a polynomial with 5000 monomials and this is why the simplest algorithm is used. Determining the asymptotic complexity of this algorithm is hard, though it is fast enough to output 25 expressions long and can sometimes fill a normal screen in under a second.

Exponentiation uses the fast method employing powers of 2. The multiplications for a given power *K* is *O(log K)*.

Description of the generation algorithm:

1. The user fills in a Descriptor from the website (currently there are three types - Equation, Inequation and Experssion)
2. Depending on the type of problem selected, a function is called in Interface.hpp. Every such function has the same format: an input of the corresponding Descriptor and a number of problems to generate, and an output consisting of two strings, for the problem and the answers respectively.

Furthermore, let's discuss the generation of an equation. First, we need some parameters. Those are supplied by EquationDescriptor and regulate various aspects of the result – coefficients, power of the equation, number of solutions, complexity and more. In the beginning, the program builds a "base" equation. For example, if we have power=3, the program will end up with something like this: (x+1)(x+2)(x-4)=0. Then it would compute the expression, so the answers aren't obvious. It would then proceed to "obfuscate" the equation by adding additional components. For instance, if we have x^2-4 = 0 at some step, after adding such a component it will become 2x^2-6x+5=(x-3)^2. The number of times this step is repeated is controlled by the descriptor. In the end, the program balances the resulting expression in such a way, that an approximately equal amount of characters is on either side of the symbol for equality. For inequations, the only difference is that we use a different sign.

Here is a method of the **Inequation** class:

```
    bool nice;
    CoefDescriptor cf, trCf;

    RNJ rnj;

    void create(InequationDescriptor& id)
    {
        letter = id.letter;

        Number root = rnj.nextNumber(id.rd);
        sign = rnj.nextInt(0, 3);

        solution = Interval(root, sign);

        nice = id.nice;
        cf = id.cf;
        trCf = id.transformCF;

        if(id.nice)
        {
            left = Expression(Number(root.fraction.down), letter);
            right = Expression(Number(root.fraction.up));
        }
        else
        {
            left = Expression(Number(1), letter);
            right = Expression(root);
        }

    }

    void addTerm(int power)
    {
        Term t = rnj.nextTerm(cf, power, letter, nice, letter);
        t.coef = rnj.nextNumber(trCf);
```

```
1:**-  Inequation.hpp   22% L35   Git-7kl  (C++/l Abbrev)
```

Here is the algorithm which generates equivalent expressions:

1. Initially, create two empty expressions (Expression), answer and solution respectively.
2. First we determine the number of components to add. This is controlled by the minTerms and maxTerms members of the descriptor.
3. Then we decide how many brackets the component should have.
4. Generate polynomials to fill in the brackets.
5. All of the polynomials generated are multiplied together in a separate polynomial. By doing this, we have two expressions that are equal to each other - one of them is in the bracket format, the other one is computed.
6. In the end, the same component is added to both expressions. We added to one of the expressions it in bracket format, and to the other – in computed format.

3 Results and Discussion

The application consists of several parts, the main ones being the core and the user interface.

The relationship between the core and the site ("Web UI") is carried out by **LuaJIT** [6] and **uWSGI** [7], to be as fast as possible, and because there **LuaJIT FFI** (Foreign function interface), which allows it to connect the core with the web server. ReactJS library [8] is used to create the user interface which made the creation of the design considerably easier. We implemented **json**, because it is a modern industry standard, and it is very easy to use.

The connection with the site, Figure 1, is accomplished in the following way: the library receives the task type and its describer from the site.

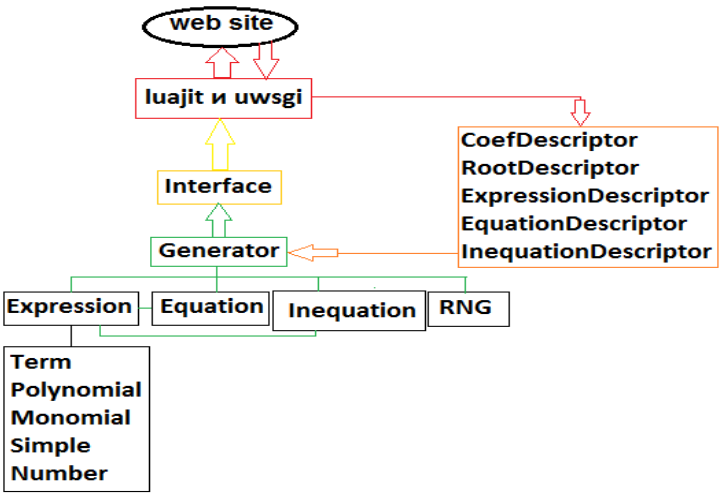

Figure 1

The library returns two strings – the first one is for the problem and second – for the answer. These strings are rendered beautifully by KaTeX JavaScript library which is the fastest drawing of LaTeX. Our web application is available at http://math4all-scholars.rhcloud.com/.

4 Conclusion

In conclusion, we created an interactive web application which is used by math teachers and students. They think that this is functional and useful and teachers often generate worksheets for classes and exercises with "Math for all". Our goal is to help both teachers and students by giving them a powerful tool to design expressions and equations with particular properties and features. In the future we intend to expand our work in order to add more functions to the generator so as to create some power inequalities.

Acknowledgements
This work is partially supported by Net Create which provided free access to their web servers.

References

[1] S. Nakov, P. Dobrikov, Programming = ++ Algorithms, 2nd edition, *TopTeam Co.*, Sofia 2003.
[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 2nd Ed., 2001.
[3] J.M.de Koninck, A. Mercier, 1001 Problems in Classical Number Theory, American Mathematical Society, 2007.
[4] S. Petkova, J. Ninova, S. Matakieva, Mathematics for 7th grade, *Prosveta Publishing*, 2013.
[5] https://www.sgi.com/tech/stl/index.html
[6] http://luajit.org/
[7] http://uwsgi-docs.readthedocs.org/en/latest/Lua.html
[8] https://facebook.github.io/react/