ЛИТЕРАТУРА

1. BOOTSTRAP VS MATERIAL COMPARISON [Электронный ресурс] – Режим доступа: https://www.youtube.com/watch?v=CgjlftJNfKc&ab_channel=Jelvix (дата обращения 16.04.2022)

2. Bootstrap vs. Material-UI. Which One to Use for the Next Web App? [Электронный ресурс] – Режим доступа: https://flat-logic.com/blog/bootstrap-vs-material-ui-which-one-to-use-for-the-next-web-app/ (дата обращения 16.04.2022)

3. Bootstrap vs. Material: Which is the True Frontend Maestro? [Электронный ресурс] – Режим доступа: https://www.sim-form.com/blog/bootstrap-vs-material/ (дата обращения 16.04.2022)

УДК [004.056+003.26](075.8)

Студ. С.И. Тумаш
Науч. рук. проф. П.П. Урбанович
(кафедра информационных систем и технологий, БГТУ)

ГЕНЕРАЦИЯ ЭФФЕКТИВНОЙ ХЕШ-ФУНКЦИИ МЕТОДОМ ПЕРЕБОРА

Хеш-функции нашли применение в решении таких задач, как реализация ассоциативных массивов, вычисление контрольных сумм данных для последующего обнаружения в них ошибок, хранение паролей, выработка электронной подписи и др. [1].

Задачей нашего исследования является разработка случайного генератора хеш-функций и замера эффективности функций, полученных в результате его работы, что позволило бы в дальнейшем выявить некоторые закономерности, объединяющие наиболее эффективные из сгенерированных.

В качестве характеристики, определяющей эффективность, выбирается лавинный критерий [1].

Определим генератор как пару $G = (O, k)$, где $O = \{o_1(x),\ o_2(x), \dots, o_n(x)\}$ – множество элементарных операций, k – количество случайно выбираемых из O операций, из композиции которых будет состоять *H(x)* [2, 3].

Тогда сгенерированная им хеш-функция $H(x)$, представляющая собой композицию случайно выбранных операций из O, определяется следующим образом: $H(x) = r_1(x) \circ r_2(x) \circ \dots \circ r_k(x)$, где $r_i(x) \epsilon_R O$, т. е. случайно выбранная, согласно дискретному равномерному распределению, функция из множества операций *O*.

Целесообразно в качестве элементарных операций *O* использовать обратимые (биективные) функции, т.к. они взаимно однозначно отображают множество входных значение на множество выходных, что в мире хеш-функций означает отсутствие коллизий.

Докажем, что результирующая функция *H(x)* в таком случае будет обратимой. Пусть *f* и *g*–обратимые функции, f^{-1} и g^{-1}– обратные им функции соответственно, а *h*–композиция *f* и *g*. Зная, что композиция ассоциативна, запишем:

$h \circ (g^{-1} \circ f^{-1}) = (f \circ g) \circ (g^{-1} \circ f^{-1}) = (f \circ f^{-1}) \circ (g \circ g^{-1}) = Id \circ Id = Id$,

где *Id*–тождественное отображение, т. е. функция, переводящая аргумент в себя. Таким образом, *h* также является обратимой, и обратной для нее является $(g^{-1} \circ f^{-1})$.

В рамках данной работы с целью избежать излишне комплексных объяснений, будет рассматриваться и генерироваться лишь подмножество целочисленных функций, т. е. принимающих и возвращающих целое число. Но и все целые числа мы рассматривать по понятным причинам не будем, лишь те, что представлены структурой System.Int32 из BCL– части CLI, разработанной Microsoft.

Набор операций, используемых в программной реализации:

1. $o_1(x) = \sim x$
2. $o_2(x) = x + const$
3. $o_3(x) = x - const$
4. $o_4(x) = x \wedge const$
5. $o_5(x) = x \wedge (x \gg const)$
6. $o_6(x) = x \wedge (x \ll const)$
7. $o_7(x) = x - (x \ll const)$
8. $o_8(x) = x + (x \ll const)$

1. $o_1(x)^{-1} = o_1(x)$
2. $o_2(x)^{-1} = x - const$
3. $o_3(x)^{-1} = x + const$
4. $o_4(x)^{-1} = o_4(x)$
5. ${o_5}^{-1}$нетривиальна, см. обратное преобразования Кода Грея
6. ${o_6}^{-1}$ищется по аналогии с ${o_5}^{-1}$
7. $o_7(x)^{-1} = \frac{x}{1-2^{const}}$
8. $o_8(x)^{-1} = \frac{x}{1+2^{const}}$

Разумеется, это не все обратимые операции, которые имеет смысл использовать в хеш-функциях: какой-нибудь циклический сдвиг или умножение на нечетную константу тоже могут себя неплохо показать. Используемая в программной реализации методика вычисления приблизительного значения лавинного эффекта:

1. Выбрать случайное число *x*.
2. Вычислить хеш *x* с помощью сгенерированной функции.
3. Инвертировать в *x* бит номер *n*+1 (начальное значение *n* = 0).
4. Вычислить новое значение хеша измененного *x*.
5. Вычислить сумму по модулю 2 предыдущего и нового хеша *S*.
6. Вычислить количество выставленных битов в *S* и разделить это число на общее количество битов в *S* (32 для System.Int32).

7. Записать результат деления.
8. Увеличить n на 1.
9. Если $n < 32$, то перейти к третьему пункту.
10. Вычислить среднее значение результатов деления, получив искомый лавинный эффект.

Для генерации функций, их компиляции и запуска во время выполнения программы использовались классы из сборки System.Linq.Expressions, являющейся частью .NETDLR. Сборка содержит классы, позволяющие формировать из программных инструкций объекты, представляющие собой синтаксические деревья, которые можно компилировать и исполнять. Кроме того, такое удобное представление программных инструкций позволяет легко преобразовать дерево в код на любом языке программирования (рисунок 1).

```python
def method(a: int) -> int:
    a += a << 20
    a += a << 4
    a -= a << 11
    a ^= a << 2
    a -= a << 6
    a -= 653856259
    a ^= a >> 27
    return a
```

```csharp
int Method(int a)
{
    a += a << 2;
    a ^= a >> 5;
    a += 1764512281;
    a -= 1011455830;
    a -= a << 14;
    a ^= a >> 31;
    a ^= 1473526182;
    return a;
}
```

Рисунок 1 – Сгенерированной программой код на C# и Python

Пробные запуски показывают (рисунок 3), что данное программное средство способно за разумное время генерировать функций с лавинным эффектом, приближающимся к 50 процентам, то есть удовлетворяющие Лавинному критерию. Лучшие из сгенерированных функций, конечно, стоило бы изучить подробнее, учитывая, что методика расчета лавинного эффекта не гарантирует идеальной точности, но результат в любом случае заслуживает внимания.

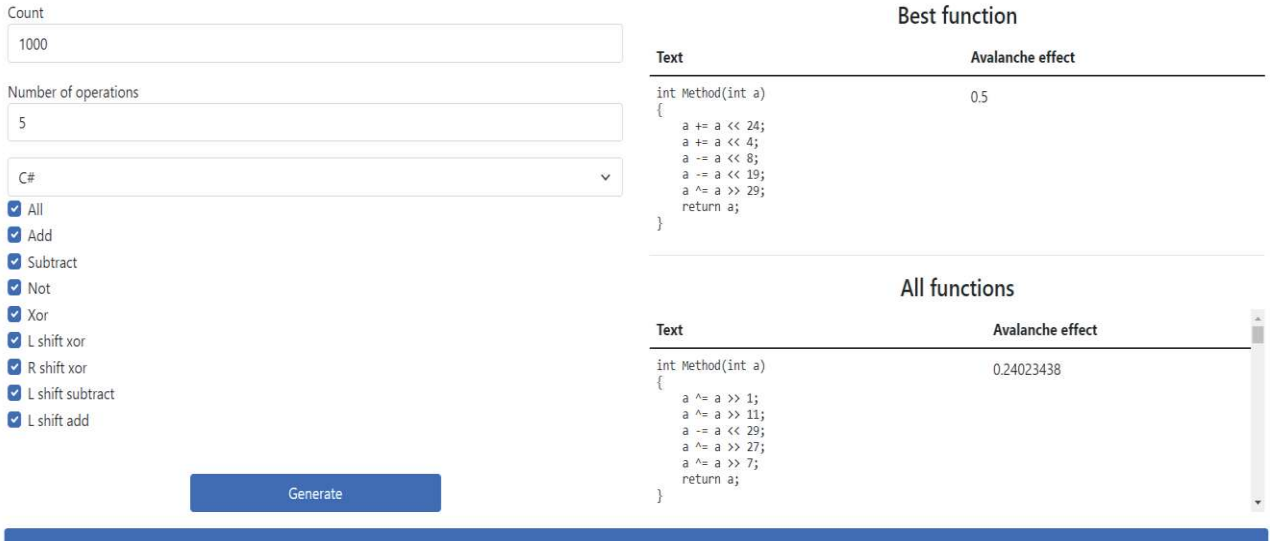

Рисунок 3 –Результат работы программы

ЛИТЕРАТУРА

1. Урбанович, П.П. Защита информации методами криптографии, стеганографии и обфускации: учеб.-метод. пособие / П.П. Урбанович. – Минск: БГТУ, 2016. – 220 с.

2. The avalanche criterion randomness test / J. Castro [et.]. // Mathematics and Computers in Simulation. – 2004. – № 6. – P. 1–7.

3. ExpressionTrees [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/expression-trees. – Дата доступа: 19.04.2022.

УДК 004.921

Студ. А.В. Кизино; Е.В. Обухова
Науч. рук. ассист. А.Н. Щербакова
(кафедра информатики и веб-дизайна, БГТУ)

ГУБКА МЕНГЕРА

Фрактал – множество, обладающее свойством самоподобия. Ковёр Серпинского – фрактал, один из двумерных аналогов множества Кантора, предложенный польским математиком Вацлавом Серпинским в 1916 г. Ковёр Серпинского представляет собой частный случай многоугольного множества Серпинского. Он состоит из 8 одинаковых частей, коэффициент подобия 1/3. Губка Менгера – трёхмерный аналог ковра Серпинского. У губки Менгера бесконечная площадь поверхности, но нулевой объем. Её размер находится между двух и трёхмерными измерениями. Сечение губки Менгера, ограниченной кубом со стороной один и центром в начале координат, плоскостью {x+y+z=0} содержит гексаграммы.

Алгоритм построения губки Менгера

1. Каждая грань куба, имеющая единичную длину, делится на 9 равных квадратиков так же, как и при построении квадратного ковра Серпинского.

2. Удаляя 7 кубиков (один центральный и 6 из центра каждой из граней), противоположные грани исходного куба соединяются сквозным центральным отверстием квадратной формы.

3. Итерационная процедура с вырезанием сквозных отверстий и последующего превращения каждого оставшегося кубика в 20 еще более мелких продолжается до бесконечности [1].

Применение губки Менгера

1. Архитектура.
2. Медицина.
3. Строительство.
4. Экология.