РАЗРАБОТКА АЛГОРИТМА ФОРМИРОВАНИЯ ВАРИАНТОВ ИГРЫ ДЛЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ "СУДОКУ"

А.М. Ширыкалов
Томский политехнический университет
ams28@tpu.ru

Введение

Сегодня, в связи с распространенностью смартфонов, большой популярностью пользуются мобильные игры – головоломки. Одним из видов таких головоломок является Судоку. В целях изучения технологии разработки мобильных приложений для операционной системы *Android* нами было выбрано создание мобильной игры Судоку.

Одной из особенностей нашего приложения является то, что в нем не используется готовый набор головоломок, а каждый раз, когда пользователь начинает новую игру, генерируется новая таблица Судоку.

В ходе разработки алгоритма генерации Судоку, возникла необходимость подсчета количества возможных решений. Это нужно было выполнить, чтобы удостовериться, что конечная таблица соответствует основному правилу Судоку (правильно составленное игровое поле должно иметь единственное решение).

Дерево решений Судоку

Далее, для упрощения объяснений, будет использована таблица 4x4. Ниже на рисунке приведен пример такой таблицы с её решением.

3			2
	4		
		1	

4	2	3	1
3	1	4	2
1	4	2	3
2	3	1	4

Рис.1. Судоку 4x4 и его решение

Хорошей визуализацией состояния нерешенного Судоку является «лес», где в каждом «дереве» на *i*-том ярусе расположены числа, подставляемые в *i*-тую свободную клетку первоначальной таблицы.

Деревья строятся по такому принципу: от каждого узла *i*-того яруса строим ветви с цифрами, которые можно по правилам поставить в *i*-тую свободную клетку изначальной таблицы, с учетом того, что в эту таблицу в предыдущие свободные клетки были поставлены цифры соответствующие пути из узла в корень дерева.

Ниже показан такой «лес» для первой таблицы с рисунка 1.

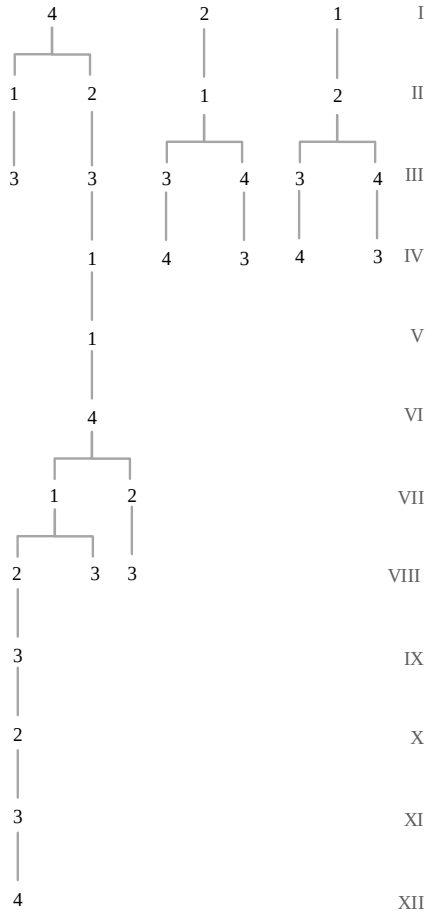

Рис.2. Полное дерево решений Судоку

Полученные таким образом леса и деревья будем называть *деревом решений Судоку*.

Алгоритм полного обхода дерева решений Судоку

Разработанный алгоритм для обхода дерева решений Судоку и подсчета количества полных решений, является рекурсивным. Он состоит из двух методов, возвращающих целые числа, рекурсивно вызываемых друг в друге.

В одном из них в таблице находится первая нулевая клетка и вызывается второй метод, с передачей ему координат клетки. В случае, если нулевой клетки не найдено, таблица считается завершенной и методом возвращается единица.

Далее приведена блок-схема работы метода нахождения первой нулевой клетки - *findZeroAndTrySolving()*.

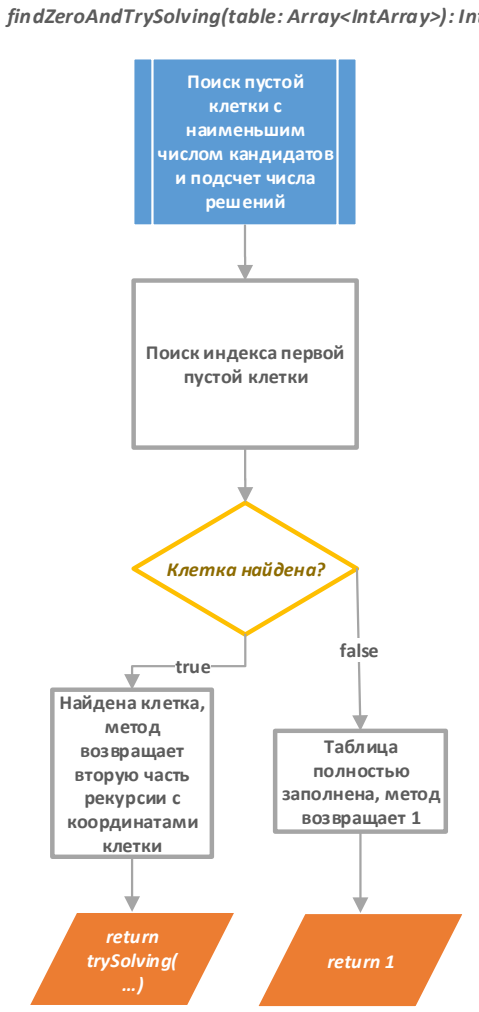

Рис.3. Блок-схема работы метода *findZeroAndTrySolving*()

Если нулевая клетка была найдена, вызывается метод trySolving(), в котором в эту клетку по очереди подставляются все возможные по правилам игры цифры, с вызовом при каждой подстановке метода findZeroAndTrySolving() с передачей в него измененной таблицы, при этом результаты суммируются. Далее приведена блок-схема его работы.

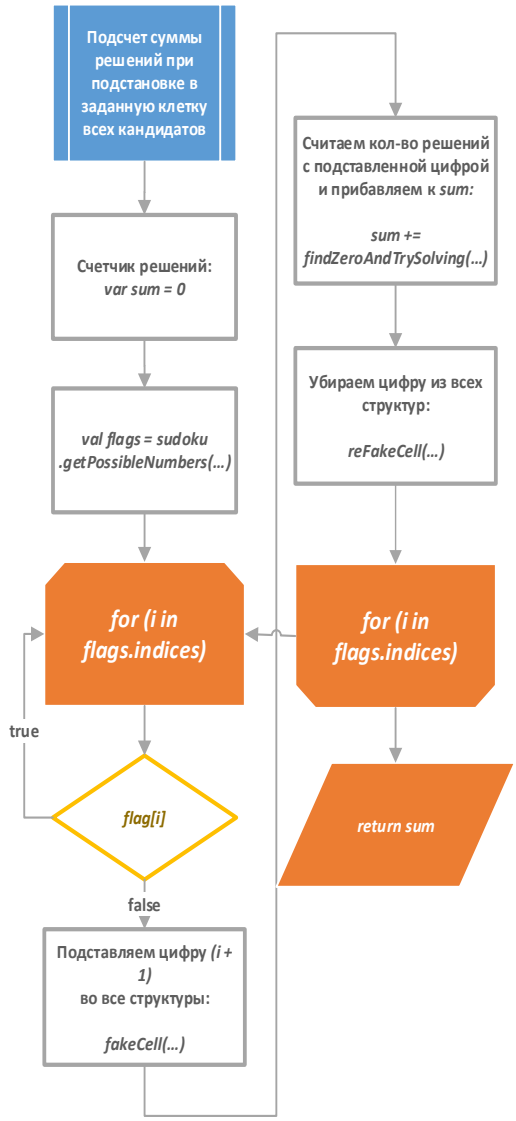

Рис. 4. Блок-схема работы метода *trySolving*()

Заключение

Как результат, в ходе работы алгоритма полностью проходится дерево решений, при этом подсчитывается количество раз, когда таблица заполнялась полностью.

Список использованных источников

1. Counting Sudoku Solution Grids using Monte Carlo. [Электронный ресурс]. – Режим доступа: https://theartofmachinery.com/2017/08/14/monte_carlo_counting_sudoku_grids.html, свободный (дата обращения: 10.08.2018).
2. Herzberg A. M., Murty R. Sudoku squares and chromatic polynomials // Notices of the AMS. – 2007. – Т. 54.– № 6. – С. 708-717.
3. Bertram F., Frazer J. Mathematics of Sudoku I. // Mathematical Spectrum. – 2006. – Т. 39 – С. 15-