Пример применимой для Proof-Of-Work функции и тестирование её применимости

Валерия Немычникова, 599 группа ФИВТ МФТИ

26 ноября 2017

1 Введение

Proof-of-Work (POW) – это принцип защиты распределённых систем от злоупотребления ими, таких, как рассылка спама. Также POW используется в некоторых блокчейн-системах, например, в Bitcoin.

Основная идея POW в том, что рассматривается некая функция, которая в одну сторону вычисляется быстро, а в другую *умеренно* сложно. Под умеренностью подразумевается (это будет формализовано позже) то, что функцию вычислить можно за разумное, но достаточно большое время. Например, если проезд по платной дороге стоит какое-то количество денег, то заработать эти деньги занимает разумное время, а проверка факта оплаты мгновенная.

В этой работе будет рассмотрен пример "хорошей"для POW функции, устроенной на основе поиска системы общих представителей данного семейства множеств, которая, по информации автора, раньше не упоминалась в статьях по этой теме. Будет приведено доказательство того, что функция удовлетворяет требуемым свойствам и приведены расчёты, показывающие её практическую применимость. В качестве демонстрации также будет приведён код простого блокчейна, который в качестве POW использует эту функцию.

Далее работа устроена следующим образом. Вначале будут даны необходимые определения. Далее будет доказано, что функция на основе с.о.п. удовлетворяет требованиям, необходимым для POW. После этого будут приведены расчёты подходящих параметров для практического использования этой функции. Наконец, будет приведено описание простейшего блокчейна, использующего вычисление системы общих представителей в качестве доказательства работы.

2 Определения

Функция f называется *pricing-функцией*, если выполнены следующие условия:

1. f умеренно сложно вычислить;
2. f не поддаётся амортизации: если даны l значений $m_1, ..., m_l$, то амортизированная стоимость вычисления $f(i) \forall i : 1 \leq i \leq l$ сравнима со стоимостью вычисления $f(m_i)$.
3. Если даны x, y, то легко вычислить, что $y = f(x)$.

Здесь под умеренной сложностью вычисления имеется в виду, что выполнение этой функции занимает существенное время по сравнению с проверкой ответа.

Пусть дано множество A и семейство его подмножеств $\mathcal{M}$. *Системой общих представителей* этого семейства называется такое $B \subset A$, что $\forall C \in \mathcal{M} \exists b \in B | b \in C$.

Минимальной системой общих представителей семейства подмножеств назовём систему общих представителей этого семейства минимального размера.

3 Pricing-функция на основе систем общих представителей

Нахождение минимальной системы общих представителей – задача, в общем случае требующая полного перебора. Известны "жадные"верхние оценки на размер системы общих представителей.

Рассмотрим множество A из n элементов. Зафиксируем k – количество элементов в его подмножестве. Рассмотрим семейства, состоящие из s k-элементных подмножеств A. Зафиксируем какое-нибудь такое семейство. Обозначим за $\zeta(n, s, k)$ размер минимальной с.о.п. для этого семейства. С одной стороны,

$$\zeta(n, s, k) \leq G(n, s, k) := \max\{\frac{n}{k}, \frac{n}{k} \ln \frac{sk}{n}\} + \frac{n}{k} + 1$$

. С другой стороны, это общая оценка, которая получается жадным алгоритмом, и в частном случае эту оценку можно улучшить, непосредственно проделав шаги жадного алгоритма, причём это займёт меньше, чем $O(G(n, s, k)) = O(n \log k)$ шагов. Сконструируем теперь pricing-функцию на основе поиска "хорошей"с.о.п. Понятно, что поиск минимальной с.о.п. не подходит, потому что невозможно доказать, что предъявленная с.о.п. является минимальной (кроме тех случаев, когда её размер совпадает с нижней оценкой). Поэтому введём такую функцию:

$$F_{n,s,k}(\mathcal{A})$$

– это такое число, что оно меньше жадной оценки, и существует с.о.п. такого размера. .

Определим F только для такого семейства подмножеств n – элементного множества, что каждый элемент n-элементного множества встречается хотя бы в одном множестве семейства. [в окончательной версии будет формальное определение функции]

Покажем, что F является pricing-функцией.

1. F (по крайней мере, как известно на данный момент) можно вычислить только полным перебором, который займёт $O(2^{2^n})$.
2. Если n, s, k фиксированы, то неизвестно такого метода, который бы позволил получить с.о.п. какого-либо семейства множеств, зная с.о.п. другого семейства множеств, быстрее, чем полным перебором. Известные рекурсивные оценки используют меньшие значения параметров, а не равные.
3. Зная с.о.п. в явном виде, можно за $O(n^2 \log k)$ непосредственно проверить, что такое семейство множеств является с.о.п. Это значительно быстрее, чем полный перебор из п.1.

4 Подходящие параметры для F

Подберём параметры n, k, s таким образом, чтобы перебор занимал значительное время. Будем отталкиваться от того, чтобы программа на C++ (который достаточно быстрый) работала 10 секунд. Так как основную часть работы программы будут составлять вычисления, можно считать, что программа на Python + numpy будет работать примерно столько же. Кроме времени работы ещё хочется, чтобы общее количество возможных вычислений было довольно велико. А именно, чтобы задавшись один раз тройкой параметров, можно было запустить такой блокчейн, которому хватит возможных входных данных на разумное время. Зададимся для примера одним блоком в 10 секунд. Тогда если считать, что блокчейн проработает 50 лет, ему понадобится порядка 200 миллионов различных семейств множеств для поиска в них с.о.п. Покажем, что такие параметры существуют. Заметим, что временя работы зависит от s и n, но не зависит от k, при этом при данном n $\binom{n}{k}$ максимально, когда $n = 2k$. Поэтому будем считать, что $n = 2k$. Время работы полного перебора составляет $O(n^2 2^s)$. Для подсчёта подходящих n и s был реализован скрипт на Python 3 (использовался numpy для увеличения скорости вычислений), который делает следующее: генерирует s случайных $(0, 1)$-векторов, генерирует все подмножества номеров этих векторов и проверяет, что набор векторов с такими номерами образует систему общих представителей. Как только какая-нибудь с.о.п. найдена, программа завершается. Было протестировано среднее время работы скрипта для $(s, n) = (50, 400), (50, 500)$. Среднее время работы для первого случая составило порядка 8 секунд, для второго порядка 35 секунд. [В окончательной версии будет протестировано значительно больше значений s, n и найден оптимум]. Отметим, что разных семейств подмножеств в случае $s = 50, n = 400$ будет $\binom{\binom{400}{200}}{50}$, а это громадное число, значительно большее 200 миллионов.

5 Описание простейшего блокчейна, использующего F

Описание появится здесь после написания программы.