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 *cost-функцией*, если выполнены следующие условия [hashcash]:

1. f умеренно (параметризуемо) сложно вычислить;
2. Если даны x, y, то легко проверить, верно ли, 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 $\forall C \in \mathcal{M}\ \exists b \in B$ такое, что $b \in C$.

Минимальной системой общих представителей семейства подмножеств назовём минимальную по мощности систему общих представителей этого семейства множеств.

3 Пример cost-функции 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. Обозначим жадную оценку размера минимальной с.о.п. для данного семейства множеств за $Gr(\mathcal{M})$.

Обозначим через $CR(\mathcal{M})$ любую систему общих представителей данного семейства множеств.

Рассмотрим следующую функцию F. Определим F только для такого семейства подмножеств n — элементного множества, что каждый элемент n-элементного множества встречается хотя бы в одном множестве семейства.

$$F_{n,s,k}(\mathcal{A}) = \begin{cases} |CR(\mathcal{M})| \text{ такой что}(Gr(\mathcal{A}) > |CR(\mathcal{M})|), \text{ если существует такая } CR(\mathcal{M}) \\ -1 \text{ иначе} \end{cases}$$

Неформально говоря, это такой размер с.о.п. для данного семейства множеств, который улучшает жадную оценку.

Теорема. *Пусть не существует полиномиального алгоритма для вычисления минимальной системы общих представителей произвольного семейства множеств. Тогда F является cost-функцией.*

Докажем это исходя непосредственно из определения.

1. F можно вычислить полным перебором. Рассмотрим всевозможные подмножества элементов семейства, у которого ищем с.о.п. и для каждого такого подмножества проверим, что оно является с.о.п. Перебор займёт $O(2^{2^n})$.
2. 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 занимает полиномиальное время. Вначале убедимся, что каждое из множеств, входящих в предъявленную с.о.п., входит в требуемое семейство, а затем проверим, что все элементы исходного множества встречаются хоть в каком-то из множеств семейства.

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.

Рассмотрим такой эксперимент. Рассмотрим несколько различных троек значений n, k, s. Сгенерируем семейство случайных k-элементных подмножеств n-элементного множества. Сначала жадно найдём с.о.п., получим жадную оценку. Потом станем искать с.о.п. полным перебором. Возможны 3 различных исхода: что сгенерированное семество вообще не будет иметь с.о.п. (иначе говоря, мы собираем школьников на олимпиаду по русскому языку, которая включает решение задач на синтаксис, морфологию и древнерусский язык, а никто из наших школьников не знает древнерусского), что полным перебором мы получим, что жадную оценку нельзя улучшить, или что жадная оценка улучшилась.

Первый случай, очевидно, нам не подходит. Второй случай тоже плохой: в качестве доказательства, что F вычислено, мы предъявим непосредственно сами множества. Но доказательство того, что мы перебрали все подмножества, будет слишком длинным. Поэтому будем считать такой случай тоже плохим.

Цель эксперимента: найти такую тройку параметров, что почти всегда будет реализовываться третий случай: что жадная оценка улучшаема. Запустив скрипт *timing.py*, мы нашли такие тройки (выделены в таблице), что при генерации случайных семейств подмножеств с такими параметрами часто получаются такие семейства, в которых интересно искать с.о.п, перебор занимает хорошее время (10-30 секунд), а разных семейств подмножеств достаточно много.

5 Описание простейшего блокчейна, использующего F

В качестве примера использования функции с параметрами, которые мы подобрали, реализуем простой блокчейн без распределённых вычислений (для демонстрации работы функции нужно лишь проверить, что хорошо работает майнинг).

Блокчейн представляет из себя цепочку объектов, называемых блоками. Блоки содержат произвольную информацию. Для генерации следующего блока цепочки необходимо вычислить некоторую cost-функцию, в параметрах которой обязательно должно содержаться значение функции от предыдущего блока (иначе, если функция зависит лишь от номера блока, любой участник системы сможет посчитать некоторые блоки „заранее", а смысл блокчейна как раз в том, чтобы блоки генерировались не чаще, чем раз в какое-то время).

Опишем, как устроена генерация блока (майнинг) в рассматриваемом блокчейне.

Зафиксируем параметры n, s, k для F. Рассмотрим множество из n элементов, обозначим его A. Рассмотрим множество 2^A. Пронумеруем все его подмножества в лексикографическом порядке. Зафиксируем произвольный генератор псевдослучайных чисел от 1 до $|2^{2^A}|$. В качестве seed для генератора будем использовать систему общих представителей, полученную на предыдущем шаге. Сгенерированное псевдослучайное число возьмём в качестве индекса в $|2^{2^A}|$ того семейства множеств, которое требуется вычислить, чтобы сгенерировать блок. Вся генерация множеств происходит за полиномиальное (и достаточно быстрое) время, а основную часть времени генерации блока займёт поиск системы общих представителей в сгенерированном множестве.

Заметим, что описанное выше работает, только если получилось для данного конкретного семейства функций улучшить жадную оценку. Если не получилось, то доказать, что был осуществлён перебор всех множеств, не получится. В таком случае надо сгенерировать новое семейство множеств и попробовать снова. Возможность так сделать достигается подбором параметров для F. Можно просто увеличить seed на 1 и взять новое значение генератора псевдослучайных чисел. Понятно, что семейств, в которых нужно искать с.о.п., значительно больше, чем всех подмножеств какого-то одного семейства. Иначе можно было бы посчитать заранее несколько значений функции и генерировать случайное число, пока не попадётся уже вычисленное.

В качестве генератора псевдослучайных чисел используется вихрь Мерсенна.

Код [будет] приложен к тексту работы.

6 Измерения времени работы программы

Таблица 1: Время работы timing.py с разными параметрами F

n	k	s	оценка неулучшаема, из 10	нет с.о.п., из 10	среднее время успешного поиска, с
10	5	10	9	0	0.007
10	5	20	5	0	0.009
10	5	50	3	0	0.025
10	5	100	2	0	0.036
10	3	10	6	2	0.022
10	3	20	7	0	0.191
10	3	50	10	0	0.0
10	3	100	10	0	0.0
10	2	10	0	7	0.06
10	2	20	4	0	1.043
10	2	50	9	0	22.443
10	2	100	8	0	370.495
10	2	10	3	6	0.05
10	2	20	1	1	0.751
10	2	50	9	0	22.286
10	2	100	7	0	363.853
10	1	10	0	10	0.0
10	1	20	2	8	0.0
20	5	10	1	8	0.074
20	5	20	2	0	1.002
20	5	50	8	0	15.636
20	5	100	5	0	224.877
50	25	10	4	0	0.019
50	25	20	8	0	0.139
50	25	50	2	0	0.732
50	25	100	0	0	0.942
50	16	10	2	6	0.073
50	16	20	3	0	1.708
50	**16**	**50**	**0**	**0**	**26.633**
50	16	100	7	0	299.161
50	12	10	1	9	0.0
50	12	20	1	2	14.011
50	12	50	1	0	670.188
50	12	100	3	0	8247.005
100	50	10	7	1	0.03
100	50	20	3	0	0.236
100	50	50	6	0	2.112
100	50	100	10	0	0.0
100	33	10	1	9	0.0
100	33	20	5	0	6.036
100	33	50	3	0	241.905