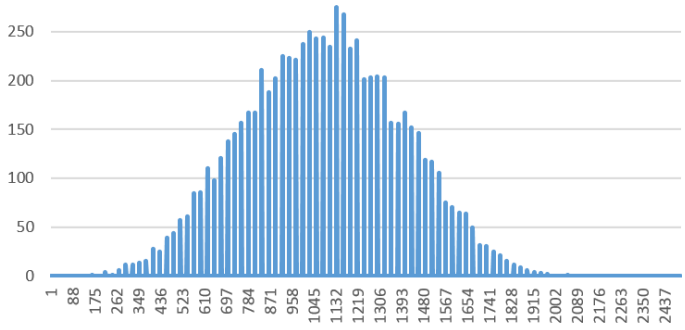
Описание хеш-функции

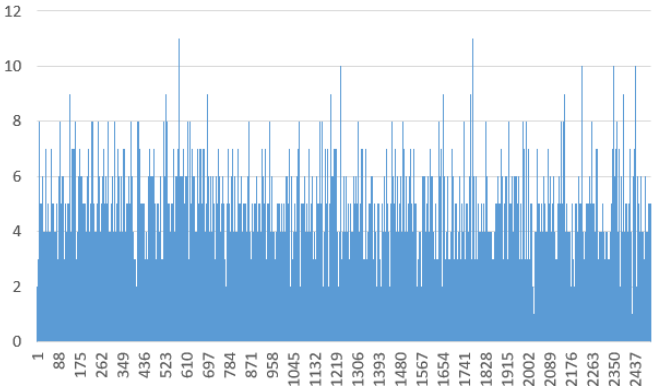
Для первой хеш-функции использовался полиномиальный хеш;
$h1 = hash(s_{0..n-1}) = s_0 + ps_1 + p^2s_2 + \ldots + p^{n-1}s_{n-1}$, где p — некоторое натуральное число , а s_i — код i-ого символа строки s.
Для второй использовалась более хеш-функция, где просто сумма квадратов кодов символов.

Результаты анализа хеш-функции

В начале я использовал самую простую хеш-функцию
h(key) = (int)key[1] + (int)key[2] + (int)key[3] + (int)key[4] +(int)key[5] + (int)key[6]
Но график попадания ключей был не равномерен, для тестов я писал для 2500 элементов вычислял 7500 хешей и смотрел куда он укажет. Возведение в квадрат каждой *key[i]* решало эту проблему, но мне так же надо было создать и вторую хеш-функцию.

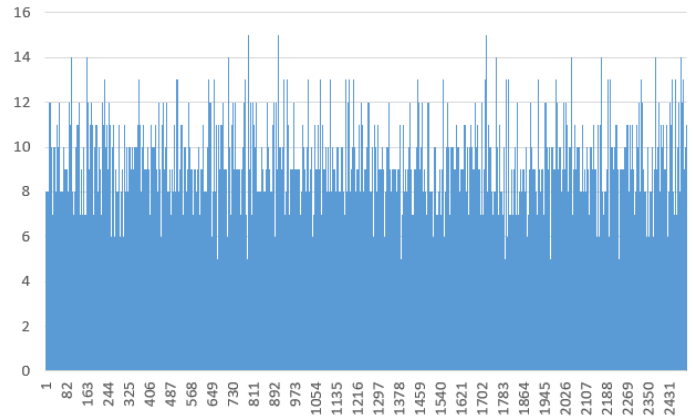


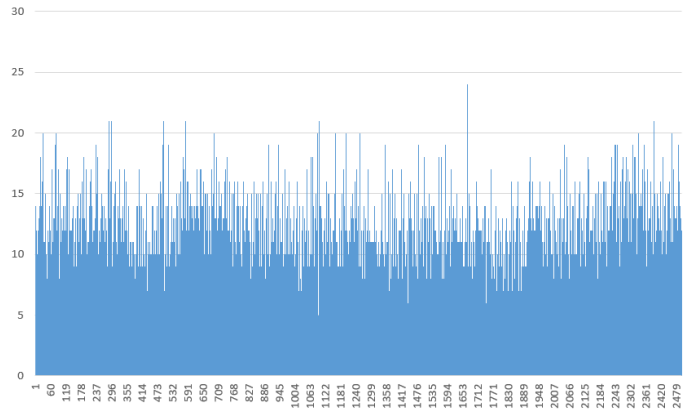
key[1] + key[2]…



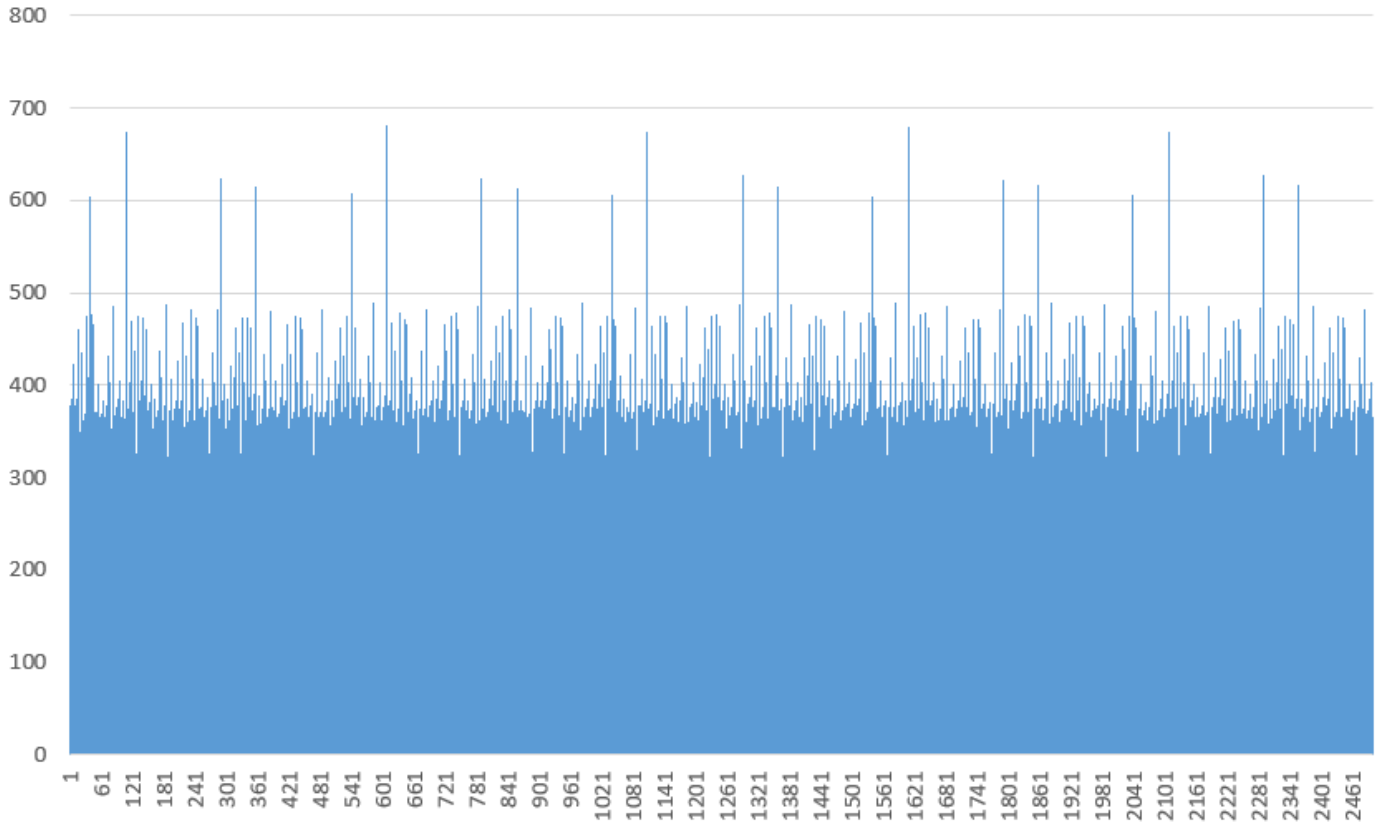
key[1]^2 + key[2] ^2…

Для первой хеш-функции я использовал полиномиальный хеш и если надо вычислять вторую, то используем key[1]^2 + key[2] ^2… для выбора простого числа. Простое число, чтобы при нахождении пустого элемента не попадать в одни и те же значения и максимальное кол-во итераций было известно заранее. Простые числа тоже вычисляются заранее до N и просто выбираются. N – число элементов в хеш-таблице. Итоговые графики кол-во обращений к элементу для заполнения всей таблицы:





Слева использование полиномиальонго хеша для, а справа использование для первой хеш-функции возведения в куб. Как видно, у первого случая в среднем пики меньше. В среднем у левого графика ~7.5, а у второго ~8.8. Так же если бы мы не использование простые числа, для h2, а просто возведение в квадрат или куб, то результат был бы таким:


800
700
600
500
400
300
200
100
0
1
61
121
181
241
301
361
421
481
541
601
661
721
781
841
901
961
1021
1081
1141
1201
1261
1321
1381
1441
1501
1561
1621
1681
1741
1801
1861
1921
1981
2041
2101
2161
2221
2281
2341
2401
2461