Анализ разных хешей при работе с хеш-таблицой

Фролов Даниил

2 апреля 2021 г.

1 Введение

1.1 Определения

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение $i = \text{hash}(key)$. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива.

Ситуация, когда для различных ключей получается одно и то же хеш-значение, называется **коллизией**. Такие события не так уж и редки — например, при вставке в хеш-таблицу размером 365 ячеек всего лишь 23 элементов вероятность коллизии уже превысит 50% (если каждый элемент может равновероятно попасть в любую ячейку). Поэтому механизм разрешения коллизий — важная составляющая любой хеш-таблицы.

1.2 Разрешение коллизий

В данной работе используется **метод цепочек**. Каждая ячейка массива H является указателем на связный список (цепочку) пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются цепочки длиной более одного элемента.

Операции поиска или удаления элемента требуют просмотра всех элементов соответствующей ему цепочки, чтобы найти в ней элемент с заданным ключом. Для добавления элемента нужно добавить элемент в конец или начало соответствующего списка.

2 Хеши

В данной работе мы будем разбирать 6 хешей.

1. Хеш, возвращающий единицу **trash hash**

```
unsigned int UselessHash (Line* string) { //trash

    return 1;

}
```

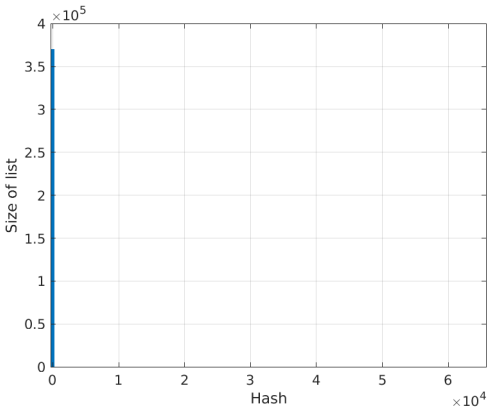

Рис. 1: trash hash

Максимально неэффективен.

2. Возвращающий длину строки **length hash**

```
unsigned int LengthHash (Line* string) { //still useless

    return (unsigned int)string->_length;

}
```

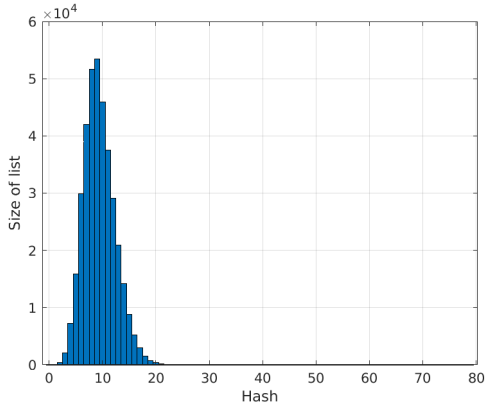

Рис. 2: length hash

Все еще максимально неэффективен, но в его оправдание, он все же лучше первого.

3. Возвращающий сумму ascii кодов символов строки **ascii summary hash**

```
unsigned int AsciiSum (Line* string) { //not so useless but so bad so still useless

    unsigned int resultHash = 0;

    for (size_t symb = 0; symb < string->_length; symb++)
        resultHash += string->_str [symb];

    return resultHash;

}
```

Вообще он уже лучше первых двух, но все еще ужасен по многим причинам.

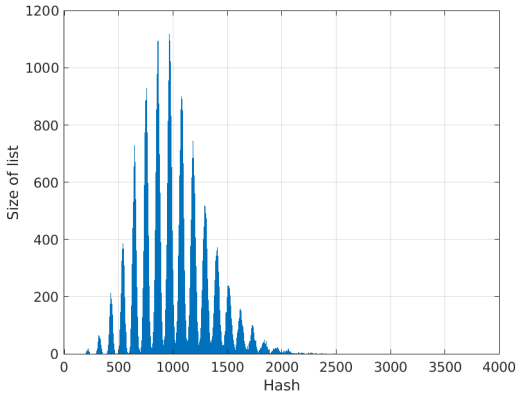

Рис. 3: ascii summary hash

4. Возвращающий первый символ строки **first symbol hash**

```
unsigned int FirstSymbHash (Line* string) { //That was not my idea, for real

    return string->_str [0];

}
```

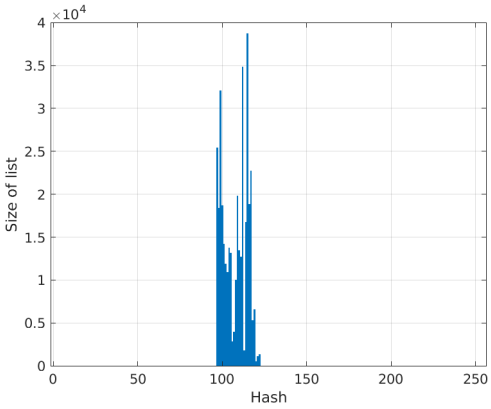

Рис. 4: first symbol hash

Данный хеш ассоциирует собой шаг назад. Вот у нас была сумма всех символов, а вот и мы смотрим только первый.

5. Хеш, построенный через ROR **ROR hash**

```
#define ROR(num) ((num << 31) | (num >> 1))

unsigned int RORHash (Line* string) { //not bad, not bad, but still not good or i am an id

    unsigned int resultHash   = 0;
    unsigned int prevSymbHash = 0;

    for (size_t symbNum = 1; symbNum < string->_length; symbNum++) {

        prevSymbHash = ROR (prevSymbHash) xor string->_str [symbNum];
        resultHash   += prevSymbHash;

    }
```

```
    return resultHash;

}
```

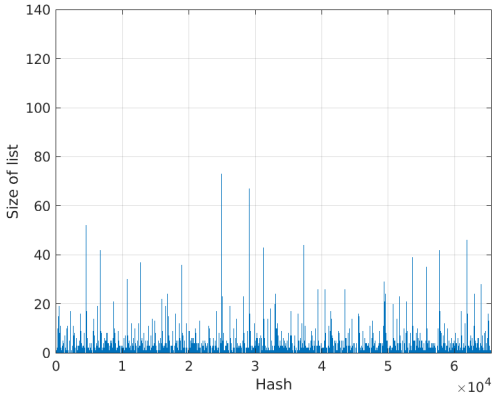

Рис. 5: ROR hash

Из всех пока перечисленных лучший по всем фронтам, но все еще большое количество коллизий: порой превосходит 80.

6. мур-мур и этим все сказано **Murmur2A hash**

```
unsigned int MurmurHash2A (Line* string) {

    const unsigned int magicConst   = 0x5bd1e995;
    const unsigned int seed         = 0;
    const          int marg         = 24;

          unsigned int hash         = seed ^ string->_length;

    const unsigned char*  data      = (const unsigned char*)string->_str;
                  size_t len        = string->_length;

          unsigned int    symb      = 0;

    while (len >= 4) {

        symb  = data [0];
        symb |= data [1] << 8;
        symb |= data [2] << 16;
        symb |= data [3] << 24;

        symb *= magicConst;
        symb ^= symb >> marg;
        symb *= magicConst;

        hash *= magicConst;
        hash ^= symb;

        data += 4;
        len  -= 4;

    }

    switch (len) {
```

```
            case 3:
                hash ^= data [2] << 16;
            case 2:
                hash ^= data [1] << 8;
            case 1:
                hash ^= data [0];
                hash *= magicConst;

        };

        hash ^= hash >> 13;
        hash *= magicConst;
        hash ^= hash >> 15;

        return hash;

    }
```

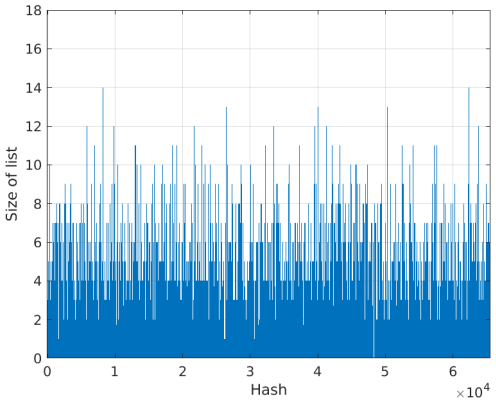

Рис. 6: Murmur2A hash

Неоспоримый чемпион из всех вышеперечисленных: мощный лавинный эффект, простота, скорость и главное, на фоне всех остальных — самое малое количество коллизий: максимальная длина списка достигла 18.

3 Вывод

Хеш-таблица — очень быстрый контейнер с самой лучше ассимптотикой, но если брать в рассчет огромное количество потенциальных коллизий, а так же абсолютно огромные затраты на память, то, естественно использовать её везде, где не попадя не нужно. Большинство приведенных хешей имеют огромное количество коллизий, из-за чего настоящая скорость удаления или поиска элемента, естественно, увеличивается и не есть good.

Лучший результат показал Murmur2A хеш. Размеры списка не превышали 18 слов.

корона

девушка слева <— Murmur2A —> девушка справа