

# Хэш функция используемая в Growthbook

- 1. Цель
- 2. Основные термины и определения
- 3. Хэш-функция
- 4. Применение соли
- 5. Всегда ли пользователям назначается один и тот же вариант эксперимента?
- 6. Симуляция хэш-функции на локальном устройстве
- 7. Дополнительно: схема процесса хэширования данных

## 1. Цель

Основная цель заключается в обеспечении прозрачности и понимания механизмов, лежащих в основе сервиса Growthbook. Понимание этих процессов важно, так как это позволяет всем заинтересованным сторонам быть уверенными в том, что результаты экспериментов надежны и представляют реальное воздействие внесенных изменений.

## 2. Основные термины и определения

**Feature Flags** — это инструмент, позволяющий управлять доступностью определённых функций (features) для разных групп пользователей.

**Хэш-функция** — это алгоритм, который принимает входные данные произвольного размера и преобразует их в выходные данные фиксированного размера.

**Соль** — строка фиксированной длины, которая добавляется (методом concat) к входным данным перед применением хэш-функции.

**Бакеты** — набор пользователей, которым показывается одна из версий продукта.

**Коллизия** — ситуация, при которой пользователи попадают в один и тот же бакет.

## 3. Хэш-функция

Хеш-функция — это математическая функция, которая преобразует входные данные (идентификатор пользователя) в некоторый набор символов. Эта "хеш-сумма" затем используется для определения, в какую группу будет распределен пользователь — группу А (контрольную) или группу В (тестовую).

Свойства хеш-функции:

- **Разнообразие** - незначительное изменение входной информации сильно изменяет хеш;
- **Равномерность** - хеш-функция должна равномерно распределять выходные хеш-значения по доступному диапазону, минимизируя количество коллизий (когда разные входные данные приводят к одинаковому хеш-значению);
- **Необратимость** - из хеша нельзя получить исходные;
- **Высокая скорость генерации**

GrowthBook использует вариацию хеш-функции FNV-1a (Fowler-Noll-Vo), называемую **hashFnv32a**, для генерации уникального хеш-значения. Это значение используется для определения того, к какому варианту эксперимента относится каждый пользователь. Функция принимает уникальный идентификатор пользователя ("value") и идентификатор эксперимента ("seed"), и на их основе создает значение от 0 до 1, которое определяет, какой вариант эксперимента будет представлен пользователю.

### Hashing

We use deterministic hashing to assign a variation to a user. We hash together the user's id and experiment key, which produces a number between 0 and 1. Each variation is assigned a range of numbers, and whichever one the user's hash value falls into will be assigned.

```
def ghash(seed: str, value: str, version: int) -> Optional[float]:
    if version == 2:
        n = fnv1a32(str(fnv1a32(seed + value)))
        return (n % 10000) / 10000
    if version == 1:
        n = fnv1a32(value + seed)
        return (n % 1000) / 1000
    return None
```

Рассмотрим подробнее параметры функции:

- **value**: Это значение, которое мы хотим захешировать. В GrowthBook это уникальный идентификатор пользователя (или иная информация), которая используется для генерации уникального хеша.
- **seed**: Это начальное значение для хеш-функции, которое помогает увеличить уникальность хеша. В GrowthBook это идентификатор эксперимента, т.е. фиче-флаг (в интерфейсе сервиса также используется термин "tracking name")
- **" " (пустая строка)**: Эта пустая строка добавляется к результату хеширования `fnv32a(fnv32a(seed + value))` как часть процесса генерации конечного хеш-значения. Иными словами, пустая строка добавляется к результату первого хеширования перед вторым применением хеш-функции (актуально только для версии 2).

**Процесс хеширования можно описать следующими шагами:**

- В версии 2 (`version == 2`), происходит двойное хеширование: сначала `seed + value`, затем результат хешируется снова с добавлением пустой строки. Результат делится на 10000 для получения значения от 0 до 1.
- В версии 1 (`version == 1`), используется одинарное хеширование `value + seed`, после чего результат делится на 1000 для получения значения от 0 до 1.

 В наших экспериментах мы используем версию 1 хэш-функции (дефолтная версия)

## 4. Применение соли

Как уже было описано выше, добавление `seed` к `value` перед хешированием (`hashFnv32a(seed + value)`) выполняет роль соли. `Seed` уникализирует процесс хеширования для каждого пользователя и каждого эксперимента.

Также, можно ошибочно предположить, что такой шаг, как добавление пустой строки используемый в версии 2 хэш функции, добавляет уникальности входных данных. Этот способ скорее про метод улучшения равномерности распределения хеш-значений, который позволяет улучшить свойства итогового хеша (равномерное и непредсказуемое распределение хеш-значений).

### **Для чего вообще нужна соль?**

В одномерных тестах, где пользователи участвуют только в одном эксперименте, соль помогает гарантировать, что хеш-значения для каждого пользователя уникальны (для каждого теста). Но ключевое применение соли приходится на многомерные эксперименты, где пользователи могут участвовать в двух или более тестах. Использование соли гарантирует, что распределение пользователей по вариантам в каждом эксперименте будет независимым и равномерным. Когда мы добавляем уникальную соль к ID пользователя для каждого эксперимента, мы предотвращаем сценарий, при котором юзер автоматически попадает в одну и ту же группу (например, всегда в контрольную) в разных тестах из-за неизменности его хеш-значения.

## 5. Всегда ли пользователям назначается один и тот же вариант эксперимента?

Пока атрибуты хеширования остаются прежним, пользователь получит тот же вариант. Как упоминалось выше, GrowthBook объединяет в хеш `hashAttribute` и `trackingKey` эксперимента, что производит десятичное число между 0 и 1. Каждой вариации присваивается диапазон значений (например, от 0 до 0.5 и от 0.5 до 1.0), и пользователь назначается той вариации, в которую попадает его хеш. При изменении настроек эксперимента некоторые пользователи могут перейти из одной группы в другую. Например, если хеш пользователя 0.49 и вы измените соотношение групп с 50/50 на 40/60 (в уже запущенном фиче флаге), диапазоны изменятся на 0-0.4 и 0.4-1. Таким образом, пользователь, который раньше был в контрольной группе, теперь окажется в экспериментальной.



GrowthBook автоматически исключает из анализа пользователей, которые видели обе версии. Но чтобы избежать путаницы и обеспечить валидность эксперимента, лучше не менять настройки эксперимента после его запуска.

## 6. Симуляция хэш-функции на локальном устройстве

Для проведения простого изолированного теста вам потребуется определить минимальный набор входных данных для создания эксперимента. Вот основные параметры, которые вам нужно будет предоставить:

```
exp = Experiment(  
    key = "simple_test",  
    variations = ["test_group", "control_group"],  
    seed = "simple_seed", # опционально  
    weights = [0.5, 0.5],  
    coverage = 1,  
    hashVersion = 2  
)
```

- **key**: Уникальный ключ эксперимента, равно название feature flag. Это строковый идентификатор, который помогает отличать один эксперимент от другого.
- **variations**: Варианты (группы), между которыми будет происходить выбор.
- **seed** (опционально): Строка, добавляемая к `device_id` при хэшировании для определения группы. Если не указано, то по умолчанию используется ключ эксперимента (`key`).
- **weights**: Список весов для каждого варианта, указывающий, какая доля пользователей должна попасть в каждый вариант. Сумма весов должна быть равна 1. Если не указано, варианты будут иметь равные веса.
- **coverage**: Процент пользователей, которые будут включены в эксперимент. Значение должно быть между 0 и 1. По умолчанию включены все пользователи (1).
- **hashVersion**: Версия хэш-алгоритма, используемого для распределения пользователей по группам.

Сначала определим хэш-функцию и другие вспомогательные функции:

Реализация алгоритма хеширования FNV-1a с 32-битным хешем. Функция `fnv1a32` принимает строку в качестве входных данных и возвращает 32-битное целое число, которое является хеш-значением входной строки. Алгоритм использует начальное хеш-значение (`hval`) и простое число (`prime`) для вычисления хеша.

### fnv1a32

```
from typing import Optional, Tuple, List  
  
def fnv1a32(str: str) -> int:  
    """  
    str: Входная строка, для которой необходимо вычислить хеш-значение.  
    """  
    hval = 0x811C9DC5  
    prime = 0x01000193  
    uint32_max = 2 ** 32  
    for s in str:  
        hval = hval ^ ord(s)  
        hval = (hval * prime) % uint32_max  
    return hval
```

Функция `gbbhash` для генерации нормализованного хеш-значения на основе входных данных и версии алгоритма. В зависимости от версии, она использует разные подходы к хешированию. Возвращает нормализованное значение хеша в диапазоне от 0 до 1 или `None`, если версия указана неверно.

## gbhash

```
def gbhash(seed: str, value: str, version: int) -> Optional[float]:
    """
    seed: Строка-сид, используемая для генерации хеша, добавляется к значению перед хешированием.
    value: Строка, для которой необходимо вычислить хеш-значение.
    version: Версия алгоритма хеширования. В зависимости от версии используются разные методы хеширования.
    """
    if version == 2:
        n = fnv1a32(str(fnv1a32(seed + value)))
        return (n % 10000) / 10000
    if version == 1:
        n = fnv1a32(value + seed)
        return (n % 1000) / 1000
    return None
```

Функция **inRange** проверяет, находится ли число *n* в заданном диапазоне *range*, который представляет собой кортеж из двух чисел (начало и конец диапазона). Возвращает True, если *n* находится в диапазоне, и False в противном случае.

## inRange

```
def inRange(n: float, range: Tuple[float, float]) -> bool:
    """
    n: Числовое значение, для которого проверяется принадлежность диапазону.
    range: Кортеж из двух чисел, определяющий начало и конец диапазона.
    """
    return n >= range[0] and n < range[1]
```

Функция **getBucketRanges** предназначена для создания списка диапазонов, которые используются для определения того, какой вариант эксперимента будет применён к каждому пользователю. Каждый диапазон представляет собой кортеж из двух чисел: начала и конца диапазона, и соответствует одному варианту эксперимента. Начальная точка (**start**) каждого диапазона устанавливается равной текущему значению *cumulative*. Конечная точка (**end**) рассчитывается как *start + weight \* coverage*.

## getBucketRanges

```
def getBucketRanges(numVariations: int, coverage: float = 1, weights: List[float] = None) -> List[Tuple[float, float]]:
    """
    numVariations: Количество вариантов эксперимента.
    coverage: Процент пользователей, участвующих в эксперименте, значение от 0 до 1.
    weights: Список весов для распределения трафика между вариантами. Если не указан, варианты получают равные веса.
    """
    if coverage < 0:
        coverage = 0
    if coverage > 1:
        coverage = 1
    if weights is None:
        weights = [1 / numVariations] * numVariations # Равные веса, если веса не предоставлены
    else:
        # Нормализация весов, чтобы их сумма была равна 1
        total_weight = sum(weights)
        weights = [w / total_weight for w in weights]

    ranges = []
    cumulative = 0
    for weight in weights:
        start = cumulative
        end = start + weight * coverage # Учитываем покрытие
        ranges.append((start, end))
        cumulative = end

    return ranges
```

**chooseVariation** определяет индекс варианта эксперимента для заданного значения *n*, основываясь на списках диапазонов *ranges*. Возвращает индекс выбранного варианта или -1, если ни один диапазон не подходит.

### chooseVariation




```
def chooseVariation(n: float, ranges: List[Tuple[float, float]]) -> int:
    """
    n: Нормализованное хеш-значение пользователя, используемое для определения его варианта эксперимента.
    ranges: Список диапазонов, каждый из которых соответствует одному варианту эксперимента. Диапазон определяется
    кортежем из двух чисел (начало и конец диапазона).
    """
    for i, r in enumerate(ranges):
        if inRange(n, r):
            return i
    return -1 # функция возвращает -1, если ни один диапазон не подходит
```

Запустим код на реальных данных по ранее проведенным тестам, используя указанные функции хеширования и распределения. Наша цель — **подтвердить, что данные функции могут воспроизвести уже известное распределение пользователей по группам теста**, основываясь на их идентификаторах и других параметрах теста.

Возьмем случайных 4 пользователей, состоявших в разных группах в тесте [\[ДЭ\] Автоматическое обновление стартовой](#).

### Пользователи / группа теста

```
SELECT event_date_msk, device_id,
       event_json -> 'group' AS "group"
FROM appmetrica.fct_ods_appm_export_events
WHERE event_date_msk between '2024-02-02' AND '2024-02-10'
      AND event_name = 'experiment'
      AND device_id IS NOT NULL
      AND event_json -> 'group' IN ('lenta_start_select_test_A', 'lenta_start_select_test_B', 'lenta_start_select_test_C',
                                     'lenta_start_select_control_D')
ORDER BY random()
LIMIT 10
```

 event_date_msk	 device_id	 group	
2024-02-08	266957EB-2792-4FA5-896D-AA935D40D0B4	lenta_start_select_test_C	
2024-02-11	51DDC532-A710-44C0-A6DB-800F2A80DBA3	lenta_start_select_control_D	
2024-02-07	0AF4BD63-83C0-4A56-B555-1F25B025F4BC	lenta_start_select_test_A	
2024-02-02	5488572A-E960-4B82-AACA-CAD01E4D3058	lenta_start_select_test_B	

В [логах фича флага "lenta"](#) мы можем найти необходимые параметры эксперимента.

Audit Log		
19		"trackingKey": "lenta",
20		"values": [
21		{
22	-	"value": "lenta_start_select_test_A",
23	-	"name": "lenta_start_select_test_A",
24	-	"weight": 0.25,
25		"id": "var_lgf7cyqe"
26		},
27		{
28	-	"value": "lenta_start_select_test_B",
29	-	"name": "lenta_start_select_test_B",
30	-	"weight": 0.25,
31		"id": "var_lgf7cyqf"
32	-	},
33	-	{
34	-	"value": "lenta_start_select_test_C",
35	-	"name": "lenta_start_select_test_C",
36	-	"weight": 0.25,
37	-	"id": "var_ls348we9"
38	-	},

Итак, мы определили ключ теста, веса групп, процент участия и другие параметры, которые были использованы в эксперименте. Теперь применим наши функции хеширования и распределения к идентификаторам пользователей с учетом настроек теста, чтобы определить, в какую группу каждый пользователь должен быть распределен:

```
key = "lenta"
variations = ["lenta_start_select_test_A ", "lenta_start_select_test_B", "lenta_start_select_test_C", "lenta_start_select_control_D"]
weights = [0.25, 0.25, 0.25, 0.25]
seed = key
coverage = 1.0
hash_version = 1

# Получение диапазонов для вариантов
ranges = getBucketRanges(len(variations), coverage, weights)

# Симуляция для набора пользователей
user_ids = ["266957EB-2792-4FA5-896D-AA935D40D0B4",
            "51DDC532-A710-44C0-A6DB-800F2A80DBA3",
            "0AF4BD63-83C0-4A56-B555-1F25B025F4BC",
            "5488572A-E960-4B82-AACA-CAD01E4D3058"]

for user_id in user_ids:
    # Вычисление хэша для пользователя
    user_hash = gbhash(seed, user_id, hashVersion)

    # Определение варианта для пользователя
    if user_hash is not None:
        variation_index = chooseVariation(user_hash, ranges)
        variation = variations[variation_index] if variation_index != -1 else "Не включен"
        print(f" {user_id}: {variation}, хэш={user_hash}")
    else:
        print(f" {user_id}: Не удалось вычислить хэш")
```

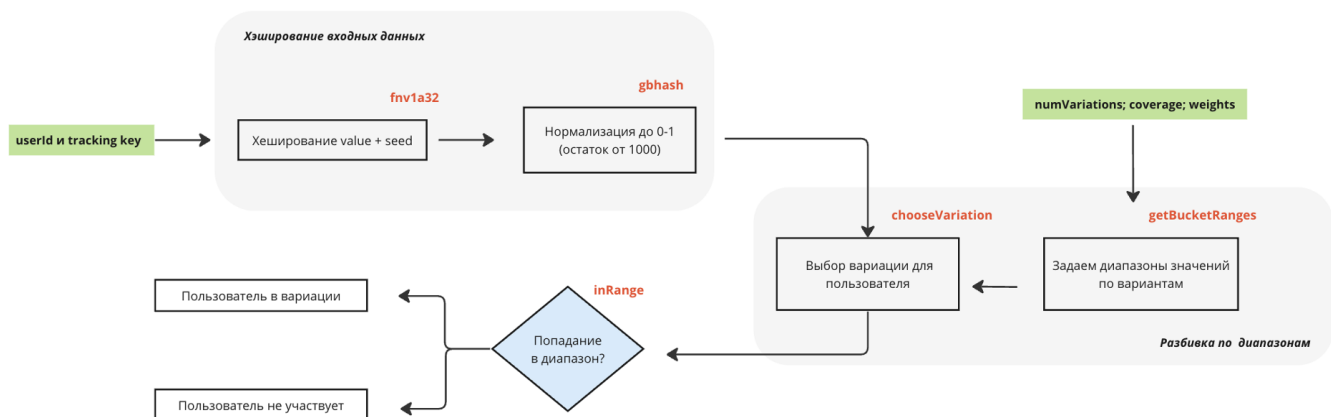
Результат:

```
266957EB-2792-4FA5-896D-AA935D40D0B4: lenta_start_select_test_C, хэш=0.735
51DDC532-A710-44C0-A6DB-800F2A80DBA3: lenta_start_select_control_D, хэш=0.884
0AF4BD63-83C0-4A56-B555-1F25B025F4BC: lenta_start_select_test_A, хэш=0.062
5488572A-E960-4B82-AACA-CAD01E4D3058: lenta_start_select_test_B, хэш=0.381
```

Сопоставив полученное распределение пользователей с историческими данными и увидев, насколько точно функции смогли воспроизвести оригинальное распределение, мы получаем несколько важных преимуществ и возможностей для дальнейшей работы:

1. **Понимание логики:** Мы четко понимаем, как алгоритм принимает входные данные, какие операции он выполняет с этими данными и как он приходит к своему конечному результату. Это означает, что мы можем объяснить каждый шаг, который алгоритм выполняет (но не всегда понимаем, почему он выполняется именно так 😊)
2. **База для масштабирования:** Успешное воспроизведение распределений на исторических данных создает основу для масштабирования наших методов на более крупные и сложные эксперименты, увеличивая нашу способность проводить множество тестов параллельно или использовать многомерные схемы.

## 7. Дополнительно: схема процесса хэширования данных



### Шаг 1 Хэширование входных данных

Сначала идентификатор пользователя и ключ эксперимента передаются в функцию **gbhash**. Происходит конкатенация value + seed после чего эта строка хешируется. Эта функция отвечает за генерацию уникального хеш-значения для каждого пользователя.

Полученное хеш-значение нормализуется к диапазону от 0 до 1 путем взятия остатка от деления на 1000 и последующего деления на 1000.

**i** Выбор числа 1000 для модуля и делителя является произвольным и служит для упрощения вычислений. Это достаточно, чтобы обеспечить хорошее распределение хеш-значений (от 0 до 999).

### Шаг 2: Разбивка по диапазонам

Используется функция **getBucketRanges** для определения диапазонов значений, соответствующих каждому варианту эксперимента. Входными параметрами являются количество вариаций (numVariations), размер трафика (coverage), которое определяет процент пользователей, участвующих в эксперименте, и веса (weights) для каждой вариации.

Функция возвращает список диапазонов (ranges), где каждый диапазон соответствует одному варианту. Диапазоны рассчитываются таким образом, чтобы учесть вес каждой вариации и размер трафика.

### Шаг 3: Выбор вариации для пользователя

С нормализованным хеш-значением пользователя и списком диапазонов вариаций, функция chooseVariation определяет, в какой диапазон попадает хеш-значение.

Если нормализованное хеш-значение пользователя попадает в какой-либо из диапазонов, функция возвращает индекс этого диапазона, который соответствует индексу вариации, в которую попадает пользователь. Если хеш-значение не попадает ни в один из диапазонов, возвращается -1, что означает, что пользователь не участвует в эксперименте.

#### **Файлы:**

[Симуляция хэш-функции.ipynb](#)

#### **Используемые источники:**

<https://docs.growthbook.io/lib/build-your-own#helper-functions>

<https://github.com/growthbook/growthbook-python/blob/main/growthbook.py>

Команда тех поддержки GrowthBook в SLACK