Хеш-функции и хеш-таблицы

Д. В. Луцив

Кафедра системого программирования СПбГУ

CS103

Содержание

Свойства

Хеш-функция:

$$h : A \to I^{(m)}$$

Обычно она опирается на *представление* элементов исходного множества.
Так как компьютер работает только с числами, машинные хеш-функции рассчитаны на представление этих чисел. В общем случае под числами можно понимать и, например, представление строк. Машинная функция обычно вычисляется от последовательности байтов.

Качество

Хорошая хеш-функция:

1. для соседних значений аргумента обычно дает сильно различающиеся результаты;
2. немонотонна и специально строится так, чтобы по её значению трудно был подобрать аргумент;
3. значения предсказуемо (обычно равномерно) распределяются по области её значений при типичных распределениях аргументов.

Эти свойства важны при использовании хеш-функций в криптографии и для защиты от повреждения информации.

Пример плохой Х/Ф

Пример плохой функции от числа или строки – несколько (например, 16) старших битов аргумента. Функция плохая потому что:

- если менять младшие биты аргумента (младшие цифры числа или хвост строки), результат вообще не изменится;
- функция монотонна;
- она будет распределена, как старшие биты аргумента. Для большого количества чисел большой длины она будет равна 0. Для большого количества строк она тоже будет распределена, в зависимости от кодировки, предсказуемо;
- в криптографии её использовать не стоит, т.к. по ней можно восстановить кусок аргумента.

Примеры

Примеры — CRC, MD5, SHA-1, SHA-2 (в порядке возрастания защищенности). У идеальной функции защищенность пропорциональна мощности пространства значений.

Использование

Необратимое шифрование паролей.

Забавный способ использования — *Proof of Work*.

▸ Перцептивный хэш — вообще не хэш, ну или ооооочень плохой, но для каких-то задач эффективен.

Использование

Необратимое шифрование паролей.

Забавный способ использования — *Proof of Work*.

Перцептивный хэш — вообще не хэш, ну или ооооочень плохой, но для каких-то задач эффективен.

Использование

Необратимое шифрование паролей.
Забавный способ использования — *Proof of Work*.
▸ Перцептивный хэш — вообще не хэш, ну или ооооочень плохой, но для каких-то задач эффективен.

Что такое CRC?

Cyclic Redundancy Check. Циклический избыточный [полиномиальный] код.

▸ В Википедии

Остаток от деления многочленов (CRC)

```
def bits2s(bits):
    return ''.join([str(k) for k in bits])

def mod2poly(dividend, divider):
    l = len(dividend)
    print(' ' * 0, bits2s(dividend), " | ", bits2s(
        divider))
    print('-' * (7 + l + len(divider)))
    while len(dividend) >= len(divider):
        bfb = bits2s(dividend)
        print(' '*(l - len(bfb)), bfb)
        headneck = [ p[0]^p[1] for p in zip(dividend,
            divider) ]
        tail = dividend[len(headneck):]
        dividend = headneck + tail
        while len(dividend) and not dividend[0]:
            dividend = dividend[1:]
    return dividend

print(bits2s(mod2poly([1,0,1,1,0,1,1,1],[1,0,1])))
```

Примеры CRC

- Стандартные

В нулевом приближении

Как и у деревьев, основное назначение — организация словарей.

Массив, который индексируется хеш-функцией ключа.

Важно: область значений х-ф обычно беднее области определения, поэтому она *делит область определения на классы эквивалентности* по признаку своего одинакового значения.

В нулевом приближении

Как и у деревьев, основное назначение — организация словарей.

Массив, который индексируется хеш-функцией ключа.

Важно: область значений х-ф обычно беднее области определения, поэтому она *делит область определения на классы эквивалентности* по признаку своего одинакового значения.

В нулевом приближении

Как и у деревьев, основное назначение — организация словарей.
Массив, который индексируется хеш-функцией ключа.
Важно: область значений х-ф обычно беднее области определения, поэтому она *делит область определения на классы эквивалентности* по признаку своего одинакового значения.

Внутреннее I

Оно же: *открытая адресация* или *замкнутое хеширование* =).
Если мы встретились со вторым элементом какого-то класса, то ячейка таблицы уже занята. Тогда мы идем по таблице далее (если требуется, с переходом через 0), пока не найдем свободную. Если таблица полна, надо её перестроить с применением более мощной хеш-функции.
При поиске мы стартуем с ячейки с номером, равным значению хеш-функции и также идем вперед, пока не найдем нужный элемент.
Если таблица не очень сильно заполнена а хеш-функция имеет подходящее распределение, то «перегруженность» классов эквивалентности маловероятна.

Внутреннее II

Как идти вперёд?

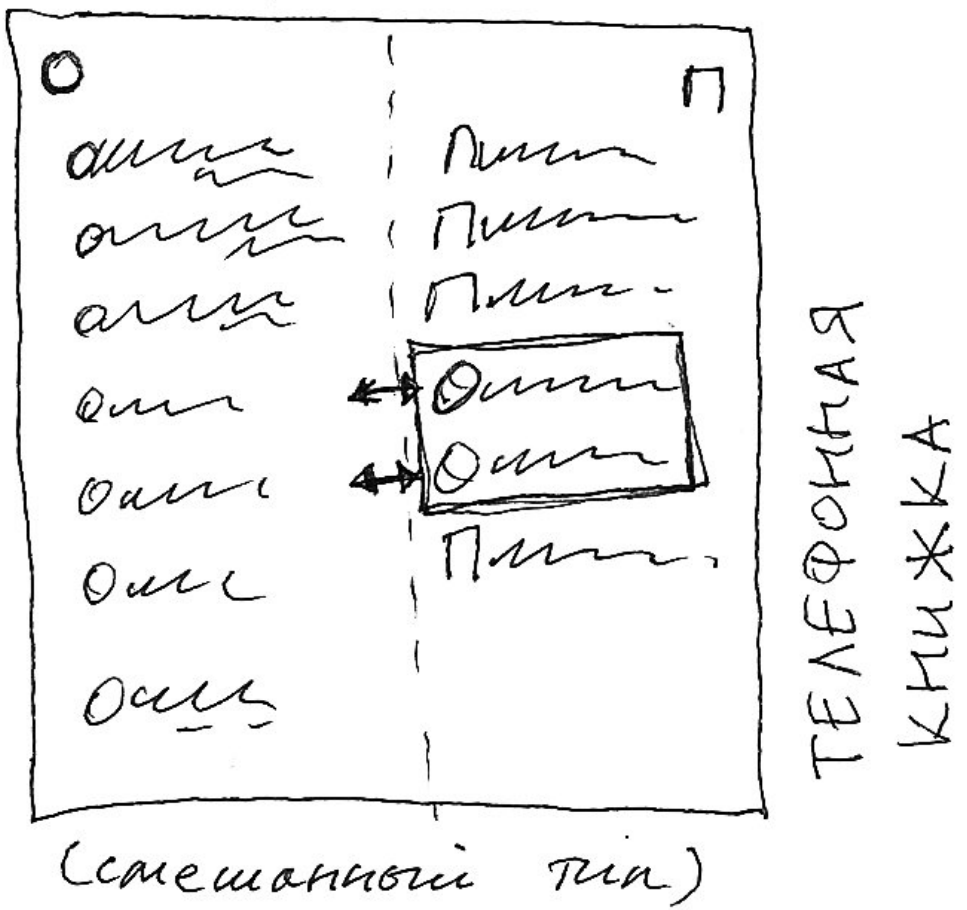
- на 1
- на константу
- повторным хешированием

Внешнее

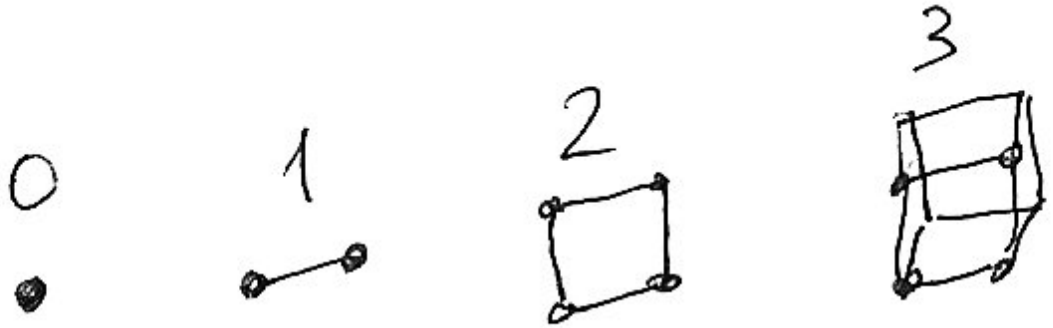
Открытое хеширование.

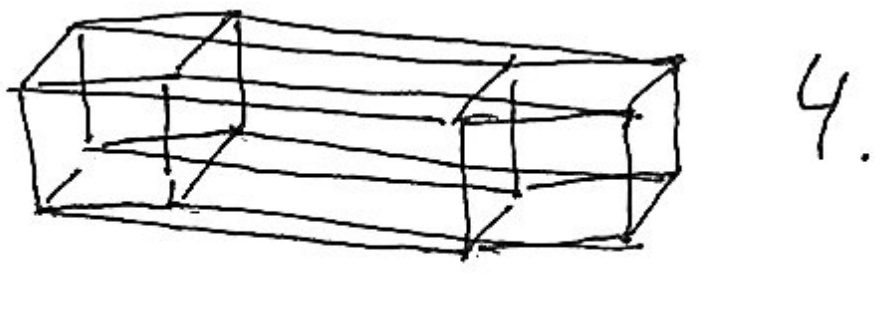
Изначально в ячейке хранится не одно значение, а список или даже дерево (!). При этом таблица может расти до бесконечности (т.к. ячейки могут расти). Но, если ячейки сильно заполнены, такую таблицу тоже стоит перестроить.

Смешанное



Гиперкуб





Топология

- Участники сети нумеруются достаточно большими случайными числами.
- Участник в первую очередь устанавливает связь с теми, от кто по номеру отличаются от него на 1 бит (метрика Кронекера).

Топология сети, таким образом, по крайней мере содержит в себе гиперкуб.

PXT существует поверх другой сети (интернета, например). И хранить все соответствия накладно. Их можно кешировать и обновлять постепенно. При запуске точки можно начать раскрутку с любого уже работающего узла: ему можно сообщить свой адрес и узнать у него адреса его соседей, и т.д., вплоть до своих соседей.

Топология

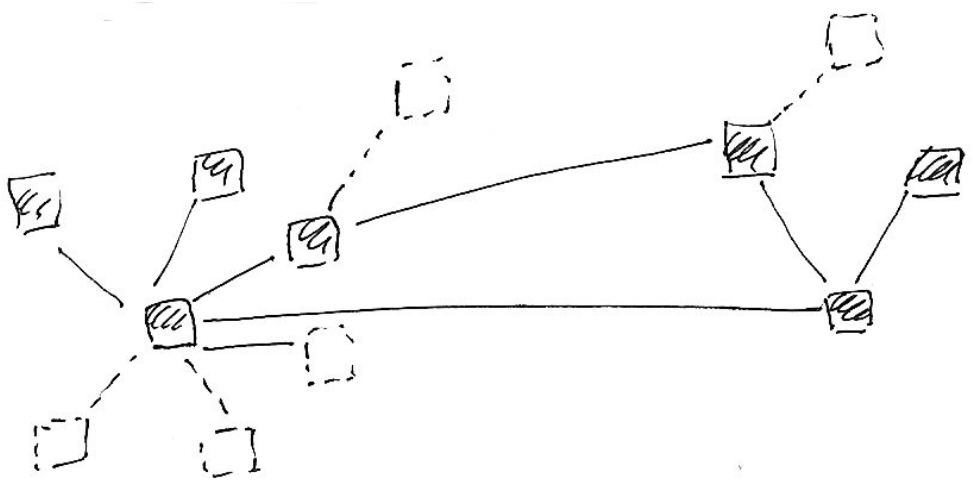
- Участники сети нумеруются достаточно большими случайными числами.
- Участник в первую очередь устанавливает связь с теми, от кто по номеру отличаются от него на 1 бит (метрика Кронекера).

Топология сети, таким образом, по крайней мере содержит в себе гиперкуб.
PXT существует поверх другой сети (интернета, например). И хранить все соответствия накладно. Их можно кешировать и обновлять постепенно. При запуске точки можно начать раскрутку с любого уже работающего узла: ему можно сообщить свой адрес и узнать у него адреса его соседей, и т.д., вплоть до своих соседей.

Хранение данных

Данные дробятся на блоки. Блоки помещаются на узлы, идентификатор которых наилучшим образом совпадает с хэш-кодом блока.

Надёжность I



1. Сеть ненадежна;
2. множество узлов сети меняется;
3. узлов на порядки меньше, чем значений хэш-функции, надо использовать только часть значения функции (например, первые биты) в качестве номера узла.

Надёжность II

Для данных с определенным кодом всегда должен существовать и быть доступен узел. Это технически обеспечить тяжело: надо 2^N узлов, где N - разрядность адреса.
Если целевой узел недоступен, работаем с ближайшими доступными вершинами гиперкуба.
Узел при запуске в первую очередь находит «похожие» узлы и скачивает с них данные, которые должны быть на нём.
Похожесть – симметричное отношение, так что можно, на всякий случай, скачать и данные соседей, чтобы, если кто-то из соседей отключится, заменить его.

Свойства

Следующие свойства распределенных хэш-таблиц делают их устойчивыми к выключению узлов и удалению части данных:

- системы децентрализованы: уже работающие узлы не нуждаются в центральных серверах, новым для запуска нужен только один работающий;
- одни и те же данные хранятся на разных машинах. При этом рекомендуется, если это возможно, «похожие» узлы выбирать с непохожими адресами несущей сети (например, с IP с разных континентов). Выключив одну машину, эти данные не уничтожить;
- на одной машине лежат данные с похожими хэш-кодами, но не куски, например, одного и того же архива. Это частично снимает с хозяина узла претензии по, например, авторским правам на его данные.

Главное думать головой

Данные по сети распространяются недетерменированно, то даже их автор не сможет уверенно удалить их.

- Если данные имеют принципиально временный характер, например, это одноразовые криптографические ключи, то это свойство не критично.
- Если же это текст, или, например, мультимедиа, то отсутствие возможности удалить данные — очередной повод подумать о том, стоит ли эти данные публиковать.

Вопросы

EDU.DLUCIV.NAME