

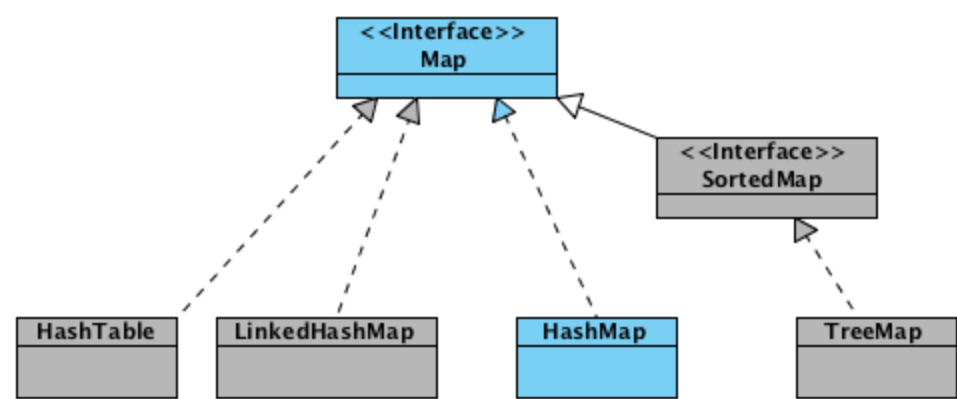
 tarzan82 5 октября 2011 в 18:07

Структуры данных в картинках. HashMap

Java*

Приветствую вас, хабрачитатели!

Продолжаю попытки визуализировать структуры данных в Java. В предыдущих сериях мы уже ознакомились с [ArrayList](#) и [LinkedList](#), сегодня же рассмотрим HashMap.



HashMap — основан на хэш-таблицах, реализует интерфейс Map (что подразумевает хранение данных в виде пар ключ/значение). Ключи и значения могут быть любых типов, в том числе и null. Данная реализация не дает гарантий относительно порядка элементов с течением времени. Разрешение коллизий осуществляется с помощью метода цепочек.

Создание объекта

```
Map<String, String> hashmap = new HashMap<String, String>();
```

Footprint{Objects=2, References=20, Primitives=[int x 3, float]}

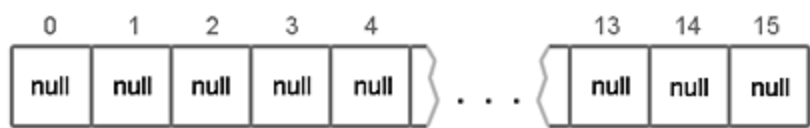
Object size: 120 bytes

Новоявленный объект hashmap, содержит ряд свойств:

- **table** — Массив типа **Entry[]**, который является хранилищем ссылок на списки (цепочки) значений;
- **loadFactor** — Коэффициент загрузки. Значение по умолчанию 0.75 является хорошим компромиссом между временем доступа и объемом хранимых данных;
- **threshold** — Предельное количество элементов, при достижении которого, размер хэш-таблицы увеличивается вдвое. Рассчитывается по формуле (**capacity * loadFactor**);
- **size** — Количество элементов HashMap-a;

В конструкторе, выполняется проверка валидности переданных параметров и установка значений в соответствующие свойства класса. Словом, ничего необычного.

```
// Инициализация хранилища в конструкторе
// capacity - по умолчанию имеет значение 16
table = new Entry[capacity];
```



Вы можете указать свои емкость и коэффициент загрузки, используя конструкторы **HashMap(capacity)** и **HashMap(capacity, loadFactor)**. Максимальная емкость, которую вы сможете установить, равна половине максимального значения **int** (1073741824).

Добавление элементов

```
hashmap.put("0", "zero");
```

Footprint{Objects=7, References=25, Primitives=[int x 10, char x 5, float]}
Object size: 232 bytes

При добавлении элемента, последовательность шагов следующая:

1. Сначала ключ проверяется на равенство null. Если это проверка вернула true, будет вызван метод **putForNullKey(value)** (вариант с добавлением null-ключа рассмотрим чуть позже).
2. Далее генерируется хэш на основе ключа. Для генерации используется метод **hash(hashCode)**, в который передается **key.hashCode()**.

```
static int hash(int h)
{
    h ^= (h >>> 20) ^ (h >>> 12);
    return h ^ (h >>> 7) ^ (h >>> 4);
}
```

Комментарий из исходников объясняет, каких результатов стоит ожидать — *метод **hash(key)** гарантирует что полученные хэш-коды, будут иметь только ограниченное количество коллизий (примерно 8, при дефолтном значении коэффицента загрузки).*

В моем случае, для ключа со значением **"0"** метод **hashCode()** вернул значение 48, в итоге:

```
h ^ (h >>> 20) ^ (h >>> 12) = 48

h ^ (h >>> 7) ^ (h >>> 4) = 51
```

3. С помощью метода **indexFor(hash, tableLength)**, определяется позиция в массиве, куда будет помещен элемент.

```
static int indexFor(int h, int length)
{
    return h & (length - 1);
}
```

При значении хэша **51** и размере таблице **16**, мы получаем индекс в массиве:

```
h & (length - 1) = 3
```

4. Теперь, зная индекс в массиве, мы получаем список (цепочку) элементов, привязанных к этой ячейке. Хэш и ключ нового элемента поочередно сравниваются с хэшами и ключами элементов из списка и, при совпадении этих параметров, значение элемента перезаписывается.

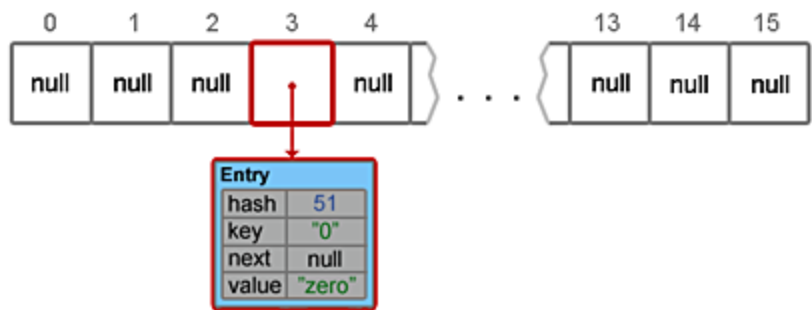
```
if (e.hash == hash && (e.key == key || key.equals(e.key)))
```

```
{
    V oldValue = e.value;
    e.value = value;

    return oldValue;
}
```

5. Если же предыдущий шаг не выявил совпадений, будет вызван метод **addEntry(hash, key, value, index)** для добавления нового элемента.

```
void addEntry(int hash, K key, V value, int index)
{
    Entry<K, V> e = table[index];
    table[index] = new Entry<K, V>(hash, key, value, e);
    ...
}
```



Для того чтобы продемонстрировать, как заполняется HashMap, добавим еще несколько элементов.

```
hashmap.put("key", "one");
```

Footprint{Objects=12, References=30, Primitives=[int x 17, char x 11, float]}

Object size: 352 bytes

1. Пропускается, ключ не равен null

2. **"key".hashCode() = 106079**

```
h ^ (h >>> 20) ^ (h >>> 12) = 106054

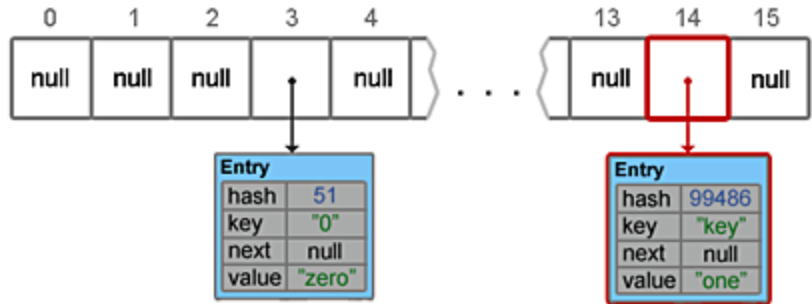
h ^ (h >>> 7) ^ (h >>> 4) = 99486
```

3. Определение позиции в массиве

```
h & (length - 1) = 14
```

4. Подобные элементы не найдены

5. Добавление элемента



```
hashmap.put(null, null);
```

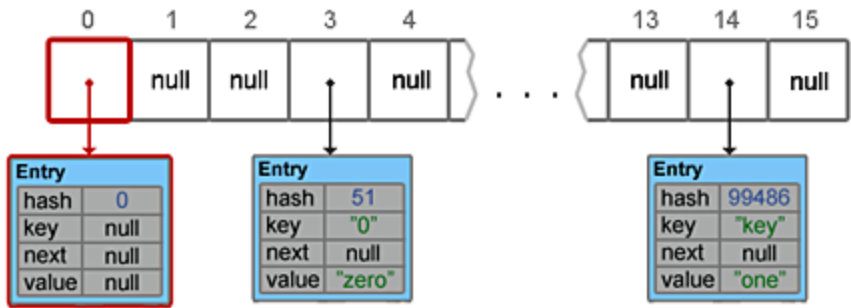
Footprint{Objects=13, References=33, Primitives=[int x 18, char x 11, float]}

Object size: 376 bytes

Как было сказано выше, если при добавлении элемента в качестве ключа был передан null, действия будут отличаться. Будет вызван метод **putForNullKey(value)**, внутри которого нет вызова методов **hash()** и **indexFor()** (потому как все элементы с null-ключами всегда помещаются в **table[0]**), но есть такие действия:

1. Все элементы цепочки, привязанные к **table[0]**, поочередно просматриваются в поисках элемента с ключом null. Если такой элемент в цепочке существует, его значение перезаписывается.
2. Если элемент с ключом null не был найден, будет вызван уже знакомый метод **addEntry()**.

```
addEntry(0, null, value, 0);
```



```
hashmap.put("idx", "two");
```

Footprint{Objects=18, References=38, Primitives=[int x 25, char x 17, float]}

Object size: 496 bytes

Теперь рассмотрим случай, когда при добавлении элемента возникает коллизия.

1. Пропускается, ключ не равен null
2. **"idx".hashCode() = 104125**

```
h ^ (h >>> 20) ^ (h >>> 12) = 104100

h ^ (h >>> 7) ^ (h >>> 4) = 101603
```

3. Определение позиции в массиве

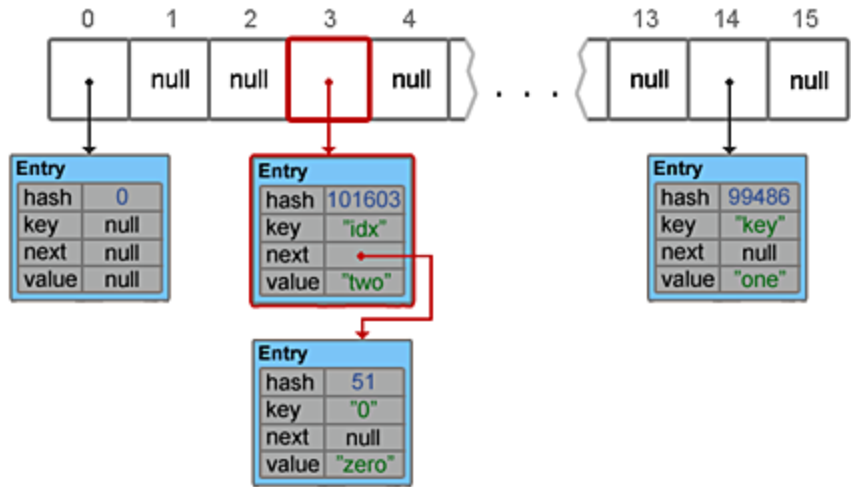
```
h & (length - 1) = 3
```

4. Подобные элементы не найдены

5. Добавление элемента

```
// В table[3] уже хранится цепочка состоящая из элемента ["0", "zero"]
Entry<K, V> e = table[index];

// Новый элемент добавляется в начало цепочки
table[index] = new Entry<K, V>(hash, key, value, e);
```



Resize и Transfer

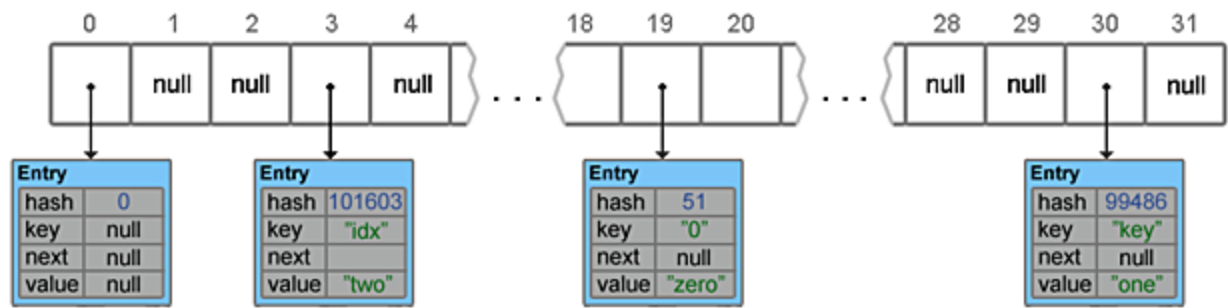
Когда массив **table[]** заполняется до предельного значения, его размер увеличивается вдвое и происходит перераспределение элементов. Как вы сами можете убедиться, ничего сложного в методах **resize(capacity)** и **transfer(newTable)** нет.

```
void resize(int newCapacity)
{
    if (table.length == MAXIMUM_CAPACITY)
    {
        threshold = Integer.MAX_VALUE;
        return;
    }

    Entry[] newTable = new Entry[newCapacity];
    transfer(newTable);
    table = newTable;
    threshold = (int)(newCapacity * loadFactor);
}
```

Метод **transfer()** перебирает все элементы текущего хранилища, пересчитывает их индексы (с учетом нового размера) и перераспределяет элементы по новому массиву.

Если в исходный **hashmap** добавить, скажем, еще 15 элементов, то в результате размер будет увеличен и распределение элементов изменится.



Удаление элементов

У HashMap есть такая же «проблема» как и у ArrayList — при удалении элементов размер массива **table[]** не уменьшается. И если в ArrayList предусмотрен метод **trimToSize()**, то в

HashMap таких методов нет (хотя, как сказал один мой коллега — "А может оно и не надо?").

Небольшой тест, для демонстрации того что написано выше. Исходный объект занимает 496 байт. Добавим, например, 150 элементов.

Footprint{Objects=768, References=1028, Primitives=[int x 1075, char x 2201, float]}}
Object size: 21064 bytes

Теперь удалим те же 150 элементов, и снова замерим.

Footprint{Objects=18, References=278, Primitives=[int x 25, char x 17, float]}}
Object size: 1456 bytes

Как видно, размер даже близко не вернулся к исходному. Если есть желание/потребность исправить ситуацию, можно, например, воспользоваться конструктором **HashMap(Map)**.

```
hashmap = new HashMap<String, String>(hashmap);
```

Footprint{Objects=18, References=38, Primitives=[int x 25, char x 17, float]}}
Object size: 496 bytes

Итераторы

Все потоки Разработка Администрирование Дизайн Менеджмент Маркетинг Научпоп 🔍

HashMap имеет встроенные итераторы, такие, что вы можете получить список всех ключей **keySet()**, всех значений **values()** или же все пары ключ/значение **entrySet()**. Ниже представлены некоторые варианты для перебора элементов:

```
// 1.
for (Map.Entry<String, String> entry: hashmap.entrySet())
    System.out.println(entry.getKey() + " = " + entry.getValue());

// 2.
for (String key: hashmap.keySet())
    System.out.println(hashmap.get(key));

// 3.
Iterator<Map.Entry<String, String>> itr = hashmap.entrySet().iterator();
while (itr.hasNext())
    System.out.println(itr.next());
```

Стоит помнить, что если в ходе работы итератора HashMap был изменен (без использования собственным методов итератора), то результат перебора элементов будет непредсказуемым.

Итоги

- Добавление элемента выполняется за время $O(1)$, потому как новые элементы вставляются в начало цепочки;
- Операции получения и удаления элемента могут выполняться за время $O(1)$, если хэш-функция равномерно распределяет элементы и отсутствуют коллизии. Среднее же время работы будет $\Theta(1 + \alpha)$, где α — коэффициент загрузки. В самом худшем случае, время выполнения может составить $\Theta(n)$ (все элементы в одной цепочке);
- Ключи и значения могут быть любых типов, в том числе и null. Для хранения примитивных типов используются соответствующие классы-оберки;
- Не синхронизирован.

Ссылки

Исходник [HashMap](#)

Инструменты для замеров — [memory-measurer](#) и [Guava](#) (Google Core Libraries).

Теги: [java](#), [HashMap](#), [структуры данных](#)


Хабы: [Java](#)

 **+69**  **1.1M**  **1734** 

Редакторский дайджест

Присылаем лучшие статьи раз в месяц

Электронпочта



113

Карма


0

Рейтинг

@tarzan82


Пользователь

Комментарии 41


- 


spiff 05.10.2011 в 19:28

Отлично! Продолжайте :)

 **+4**


Ответить




...
- 


tarzan82 05.10.2011 в 20:50

Вопрос к читателям: на ваш взгляд/опыт, оправдано ли отсутствие методов для уменьшения размера HashMap-а, и случались ли у вас из-за этого проблемы?

 **0**


Ответить




...
- 


Akvel 06.10.2011 в 05:50

имхо не нужно — если вы один раз уже достигли предела и получили новый размер, где гарантия что вы снова его не достигните. Двойная работа получится если резать.

 **+2**

Ответить




...
- 

Арх 06.10.2011 в 11:09


Иногда вообще предсказывают максимальный размер хэшмапы и просто подбирают соотв. capacity сразу, чтобы не делать дурную лишнюю работу.


Теперь по поводу статьи. А как вы итерируете мапу? =)

Я например foreach'em по MapEntry<..>. Удобно и я где то читал что это самый хороший способ.

 **0**

Ответить




...
- 


tarzan82 06.10.2011 в 12:54


На мой вкус, первый пример из статьи самый удобный и компактный. Внутри хэш-мапа, все его итераторы устроены практически одинаково.

Другой вопрос, если вам нужны только ключи или только значения, то правильнее будет юзать keySet() и values() а то получится что разработчики класса зря старались :)

 **0**

Ответить



...
- 

Арх 06.10.2011 в 13:16

Просто я раньше получал референс на ключи потом итерировался по ним доставал значения. И всё время у меня было ощущение что я жуткий рак, потому что выглядело это дело несколько


громоздко и не сильно производительно (каждый раз опять лезть в мапу).
А потом как то наткнулся на такую реализацию. Мне она показалась самой кошерной.

 0 Ответить  ...

o  **alist** 06.10.2011 в 13:29 ^

Он самый удобный, потому что следует внутреннему представлению структуры данных. Если итерироваться по ключам, то на каждом шаге придется вызывать хэш-функцию, а на это тратится время и вычислительные ресурсы.

 0 Ответить  ...

o  **Stocker** 06.10.2011 в 13:49 ^


Пока надобности не возникало, но я не понимаю почему это не реализовано. Вроде не кажется сильно сложным, а может ведь кому — то и пригодится... Вариант `hashmap = new HashMap<> (hashmap)` конечно рабочий, но я уже привык втыкать `final` везде где только можно, вроде как правильной... Я бы предпочёл иметь `trimToSize()`.

 0 Ответить  ...

o  **WarGoth** 05.10.2011 в 20:54

Спасибо за статью. Я примерно знал, как работает `HashMap`, но не задумывался как идет раскидывание по спискам хэшей и даже не догадывался о перераспределении при изменении размера

 0 Ответить  ...

o  **zEvg** 05.10.2011 в 23:42


Материал изложен изумительно. Спасибо!
Будут ли рассмотрены остальные мапы?

 +1 Ответить  ...

o  **Akvel** 06.10.2011 в 05:51 ^

А вы чем то кроме `HashMap` еще в жизни пользуетесь? :-)

 0 Ответить  ...

o  **Optik** 06.10.2011 в 08:52 ^

`LinkedHashMap` :-)

 +1 Ответить  ...

o  **alist** 06.10.2011 в 11:59 ^


О, да! На деле самая удобная коллекция. Тот факт, что пары ключ-значение идут в предсказуемом порядке, существенно облегчает дебаг. Жаль, что почти никто этой коллекцией не пользуется.

 0 Ответить  ...

o  **tagir_valeev** 09.10.2011 в 15:36 ^


Не только для дебага полезно. У нас был пример, что разработчик вычитал форму из XML в `HashMap` в виде `HashMap<String, UIControl>`, где ключ — это идентификатор контрола потом много где использовал её для быстрого получения контрола по идентификатору, но не озаботился тем, что порядок контролов не сохранился. Замена `HashMap` на `LinkedHashMap` в одном месте позволила исправить проблему, не трогая кучу готового кода.

 0 Ответить  ...

o  **randoom** 06.10.2011 в 13:19 ^

`ConcurrentHashMap`!


 0 Ответить  ...

o  **randoom** 06.10.2011 в 13:24 ^

Еще были прецеденты использования `WeakHashMap`... Довольно забавная штука если хочешь все данные хранить в одном месте и чтобы оно чистилось самостоятельно когда кому либо часть

- этих данных уже не нужна

0


Ответить
- 

maxcom

06.10.2011 в 14:42

на практике еще интересен IdentityHashMap, котором в отличие от HashMap конфликты по другому разрешаются

0


Ответить
- 

tarzan82

06.10.2011 в 09:06

Спасибо и вам!
Думаю да, мапы еще будут, правда в каком виде еще не думал.

+1


Ответить
- 

Akvel

06.10.2011 в 06:05

еще хочется дополнить иерархию, вот более полная

+4


Ответить
- 

Akvel

06.10.2011 в 06:07

Особенно вкусный ConcurrentHashMap

+1


Ответить
- 

wisd

16.04.2013 в 05:46

И чем он вкусен?

0


Ответить
- 

fealaer

06.10.2011 в 09:58

<оффтоп>
В чем нарисована диаграмма?
</оффтоп>

0


Ответить
- 

Akvel

06.10.2011 в 11:10

не в курсе, позаимствовал отсюда en.wikibooks.org/wiki/Java_Programming/Collection_Classes

0


Ответить
- 

hamMElion

01.02.2013 в 15:42

Это IBM Rational Software Architect

0


Ответить
- 

romik

06.10.2011 в 11:51

Метод hash(key) вовсе не для того, что бы «гарантировать ограниченное число коллизий», перечитайте комментарий в исходнике ещё раз. Он просто обеспечивает более равномерное распределение в случае неудачных хэш-функций.

0

Ответить
- 

MiniM

06.10.2011 в 15:01

Добавление элемента выполняется за время $O(1)$, потому как новые элементы вставляются в начало цепочки

А как на счёт проверки наличия такого ключа?

Не будет ли здесь потери от загруженности так же, как и в `get` ?

Ответить

0 Ответить

xhumanoid 06.10.2011 в 15:28

Будет, именно проверки наличия ключа даёт проблему, что вставка никогда не будет быстрее get.

Другой вопрос что в теории длинна цепочки не может быть больше $(\text{int})(\text{capacity} * \text{loadFactor})$, и для небольших мэпов это проходит. Проблемы могут случиться когда вы заполнили хеш, он разросся и это значение стало достаточно большим, потом удалили большинство и опять заполняете, может оказаться так, что все данные попадают в одну ооооочень длинную цепочку, вероятность конечно низкая, но всё может случится в этом мире.

Отдельно хотелось бы заметить нюанс, что размер таблицы для кранения цепочек всегда кратен степени двойки и при расширении удваивается, на этапе инициализации любое ваше значение округляется до ближайшей степени в большую сторону:

```
// Find a power of 2 >= initialCapacity
int capacity = 1;
while (capacity < initialCapacity) capacity <= 1;
```

Данное условие следует из оптимизации при получении текущего индекса в таблице цепочек:

```
static int indexFor(int h, int length) {
return h & (length-1);
}
```

Вот такой вот оригинальный способ получения остатка от деления $=)$ (код эквивалентен: $h \% \text{length}$)

+1 Ответить

MiniM 06.10.2011 в 15:32

Про размер таблицы и вычисление индекса мне известно — когда-то уже читал исходники.

Смущает то, что везде пишут, что добавление $O(1)$, а выборка $O(1 + a)$.
Хоть по своей сути добавление включает в себя выборку.

0 Ответить

xhumanoid 06.10.2011 в 15:58

Ну пишут далеко не везде, но в части добавления вы правы, сложность у put/get/remove одинаковая.

В статье же содержится ошибка, так как вставка в начало цепочки сама по себе хоть и гарантирует время $O(1)$, но в java перед вставкой сразу проходим по всей длинне цепочки, чтобы удостовериться что такого элемента нету, в результате $O(1)$ у нас только в идеальном случае будет когда отсутствуют коллизии.

0 Ответить

MiniM 06.10.2011 в 16:04

Я и говорю, что put содержит в себе get, т.к. помимо того, что добавляет элемент, он ещё и возвращает предыдущее значение ключа (null если такого ключа ещё не было).

0 Ответить

tarzan82 06.10.2011 в 16:59

Сделал небольшой тест — добавляются элементы с ключами от «0» до «1000000», в результате:
1) Самая длинная цепочка — 6 элементов, средняя длинна — 2 элемента
2) 2 раза было зафиксировано время > 0 при обходе цепочек, по $\sim 16\text{мс}$ каждый

Если пренебречь этими двумя проходами, то как раз получится $O(1)$.

Я пробовал добавить большее количество элементов, но памяти не хватило. Однако предварительные данные показали, по п. 2) срабатываний было больше, хотя время на каждый проход так же не превышало $\sim 16\text{мс}$.

0 Ответить

MiniM 06.10.2011 в 18:37

Ну раз этим можно пренебречь и записать просто $O(1)$, то зачем тогда для `get` писать $O(1+a)$?

Не совсем понял смысл вашего теста, что вы хотели им показать?

0 Ответить

xhumanoid 06.10.2011 в 19:06

pastebin.com/u8B5vbuk

size entry table: 65536
null: 65535
not null: 1

имею одну большую цепочку из 30к элементов

Вопрос: какая скорость работы получится?

p.s. да я знаю что это придуманный пример, но он наглядно иллюстрирует падение якобы $O(1)$, можете поставить 1кк и идти курить

+1 Ответить

1nd1go 15.04.2013 в 19:41

Я поддерживаю выше высказывшихся, т.к. $O(1)$ — не правда при вставке. Поиск на уже существующий элемент все же присутствует. Поэтому операция `get()` мало чем отличается от операции `put()`

```
318         for (Entry<K,V> e = table[indexFor(hash, table.length)];
319
320             e != null;
321             e = e.next) {
```

```
391         for (Entry<K,V> e = table[i]; e != null; e = e.next) {
392
393             Object k;
394
395             if (e.hash == hash && ((k = e.key) == key || key.equals(k)))
```

0 Ответить

maxp 18.11.2011 в 13:00

Мой плюс за статью. (кармы не хватает)

Как бы ничего нового не узнал, но все-равно впечатление хорошее.

0 Ответить

sphinks 18.11.2011 в 23:01

Спасибо за пост.

0 Ответить

ArtemDP 20.08.2014 в 19:46

Скажите, пожалуйста, как это работает?
 $h \& (length - 1) = 14$

0 Ответить

ArtemDP 20.08.2014 в 19:53

Если кому тоже интересно —
ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8#.D0.9F.D0.BE.D0.B1.D0.B8.D1.82.D0.BE.D0.B2.D0.BE.D0.B5_.D0.98_.28AND.29

0 Ответить

 **getmanartem** 16.09.2014 в 16:05

Еще можно добавить одну тонкость — код:

```
HashMap<Integer, String> hm = new HashMap<>();
hm.put(1, "1");
hm.put(2, "2");
hm.put(3, "3");
Set<Integer> s = hm.keySet();
for (Integer nextInt: s){
    if (nextInt == 3){
        hm.put(4, "4");
    }
}
```

не бросит исключения ConcurrentModificationException, потому что entry(3,«3») находится в последнем bucket`е и next(), который и генерирует собственно исключение, вызван не будет

0 Ответить

 **Sild** 17.05.2016 в 13:40

«Все элементы цепочки, привязанные к table[0], поочередно просматриваются в поисках элемента с ключом null. Если такой элемент в цепочке существует, его значение перезаписывается.

Если элемент с ключом null не был найден, будет вызван уже знакомый метод addEntry().»

Это всё в контексте добавления элемента с ключом null. Значит ли это, что в table[0] всегда храниться не больше одного элемента?

0 Ответить

 **tarzan82** 17.05.2016 в 16:36 ^

Не значит. Для некоторых ключей может получиться такой хэш, что при вызове **indexFor(hash, tableLength)** индекс будет равен 0. Так же важно не забывать, что при изменении **tableLength** распределение элементов в таблице будет другим.

0 Ответить

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

ПОХОЖИЕ ПУБЛИКАЦИИ

2 января в 12:00

LJV: Чему нас может научить визуализация структур данных в Java

+85 23K 236 11 +11

4 сентября 2021 в 16:46

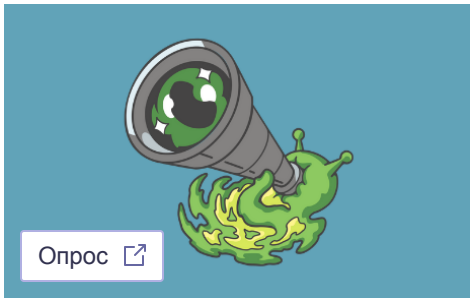
Как снизить зависимость кода от структуры данных?

+2 8.9K 59 41 +41

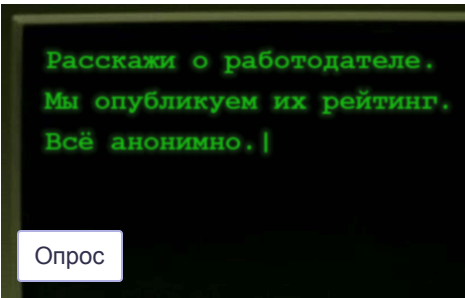
18 июля 2021 в 16:55

Две открытые библиотеки для обучения байесовских сетей и идентификации структуры данных

+6 2.3K 48 1 +1



Хотите рассказать о себе в наших социальных сетях?





Третье хабраисследование ru-IT-брендов





Тетрис на стероидах: тестируем War Robots на Steam Deck


КУРСЫ

 SQL и получение данных
16 сентября 2022 · 24 850 · Нетология

 Python для анализа данных
23 сентября 2022 · 42 000 · Нетология

 Офлайн-курс Java-разработчик
24 сентября 2022 · 34 900 · Бруноям

 Python: анализ данных и машинное обучение
3 октября 2022 · 31 000 · Loftschool

 Рассмотрение механизмов и специфики применения Генерального регламента ЕС о защите данных - General Data Protection Regulation (GDPR)
15 сентября 2022 · 35 000 · АИС

Больше курсов на Хабр Карьере

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

вчера в 20:49

Что не так с ДЭГ Москвы на этот раз?

 +259  26K  33  98 +98


вчера в 23:33

Смерть Mozilla — это смерть открытого Интернета

 +88  23K  36  211 +211

вчера в 16:03

Бифуркация (фантастический рассказ)

 +22  3K  12  13 +13


сегодня в 07:06

ЦБ хочет компенсировать страдания российских инвесторов за счет «недружественных» нерезидентов

 +21  3.9K  5  3 +3

сегодня в 00:19

Гайд по межсетевому экранированию (nftables)

 +17  4.5K  100  2 +2

Руководство по крафту: во что можно превратить статью по Data Mining

Мегапост

Ваш аккаунт

Разделы

Информация

Услуги

Войти

Регистрация

Публикации

Новости

Хабы

Компании

Авторы

Песочница

Устройство сайта

Для авторов

Для компаний

Документы

Соглашение

Конфиденциальность

Корпоративный блог

Медийная реклама

Нативные проекты

Образовательные

программы

Стартапам

Мегапроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию

© 2006–2022, Habr

ЧИТАЮТ СЕЙЧАС

Смерть Mozilla — это смерть открытого Интернета

 23K  211 +211

Что не так с ДЭГ Москвы на этот раз?

 26K  98 +98



Активность найма на IT-рынке в августе 2022

 19K  15 +15

Крякнул софт? Суши сухари

 27K  132 +132

Американские компании начали убирать кнопки Facebook** для авторизации со своих сайтов

 5.6K  7 +7

DAST ist fantastisch: отечественный динамический анализатор к взлету готов

Мегапост

РАБОТА

Java разработчик
425 вакансий

Все вакансии