Хэш-таблицы

Java Developer Level 2

Progwards - Академия компьютерного мастерства

Содержание лекции

- Хэш таблицы с прямой адресацией
- Хэш таблицы с открытой адресацией
- Виды хэш функций
- Идеальное хэширование
- Универсальное хэширование

Хэш таблица

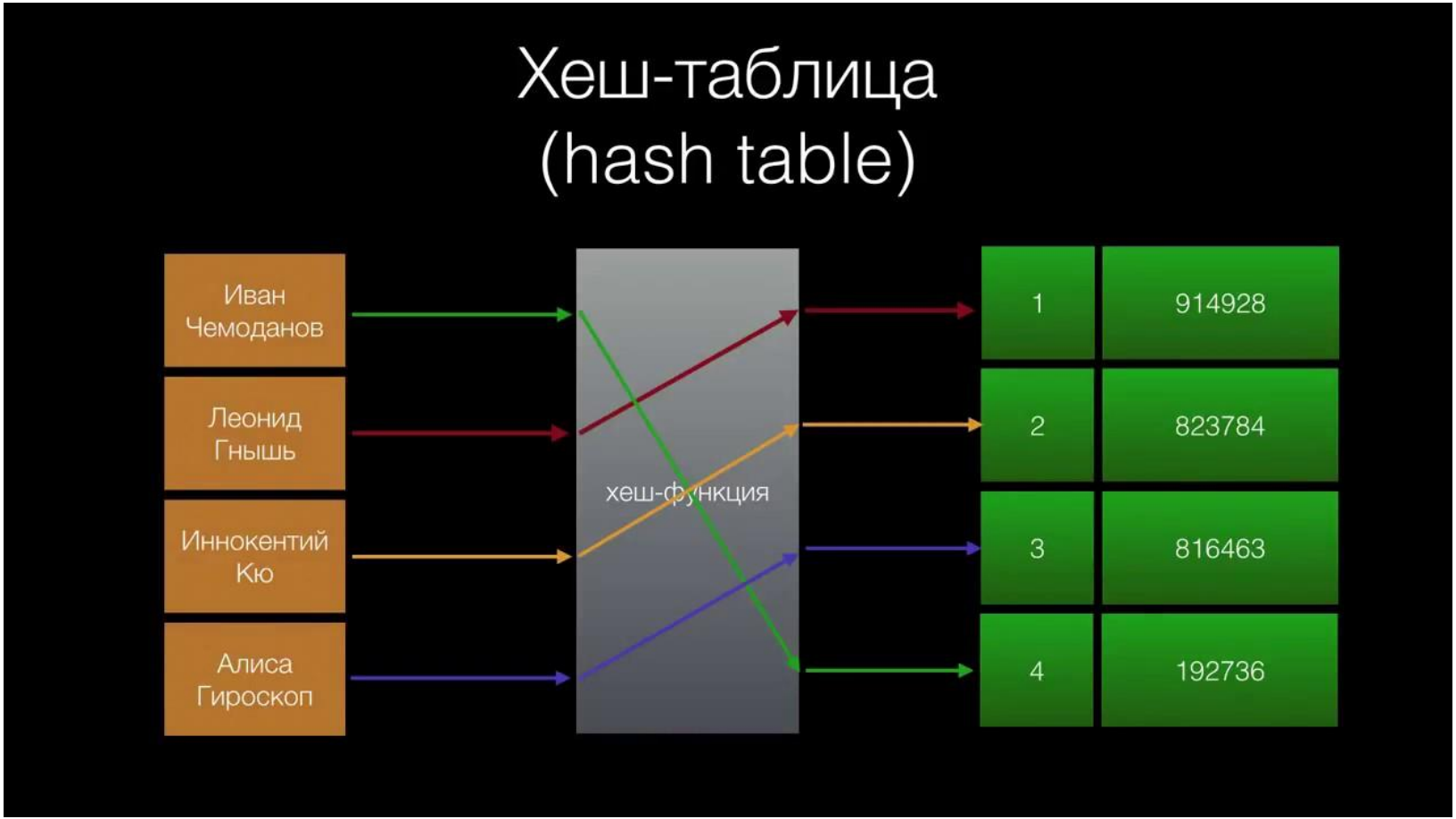
- Ассоциативный массив
- Хранит пару ключ-значение

Хэш таблица

- Добавить пару ключ-значение
- Получить элемент по ключу
- Удалить элемент по ключу

- O(1)

Хэш таблица


Хеш-таблица
(hash table)
Иван Чемоданов
Леонид Гнышь
Иннокентий Кю
Алиса Гироскоп
хеш-функция
1
914928
2
823784
3
816463
4
192736

Хэш таблица

PROGWARDS
Академия компьютерного мастерства

- Ассоциативный массив
- Хэш-функция - преобразование ключа в индекс

Хэш таблица

- Разместить массив размера size
- Хэш-функция - 0...size

Хэш таблица

PROGWARDS
Академия компьютерного мастерства

- Ключи принадлежат множеству U
- Хэш значения принадлежат множеству u
- Хэш функция это преобразование U -> u
- Совпадение значений хэш функции для разных ключей называется коллизией

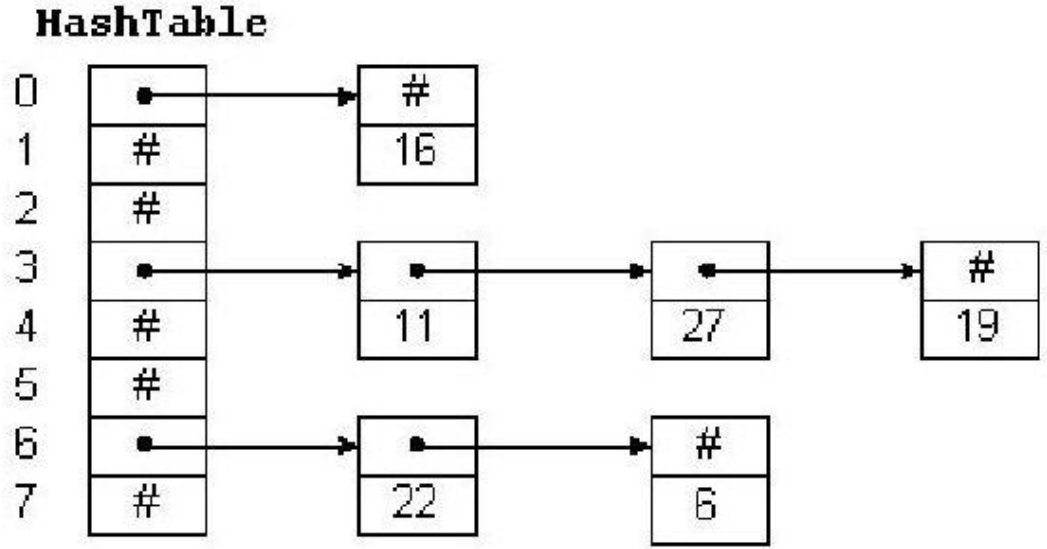
Хэш функция

- Допустим ключ - целое число
- Простая хэш-функция - остаток от деления на размер таблицы

Хэш таблица

- Ключ 246, значение 37
- 246%11 = 4

Хэш таблица с прямой адресацией



- метод цепочек

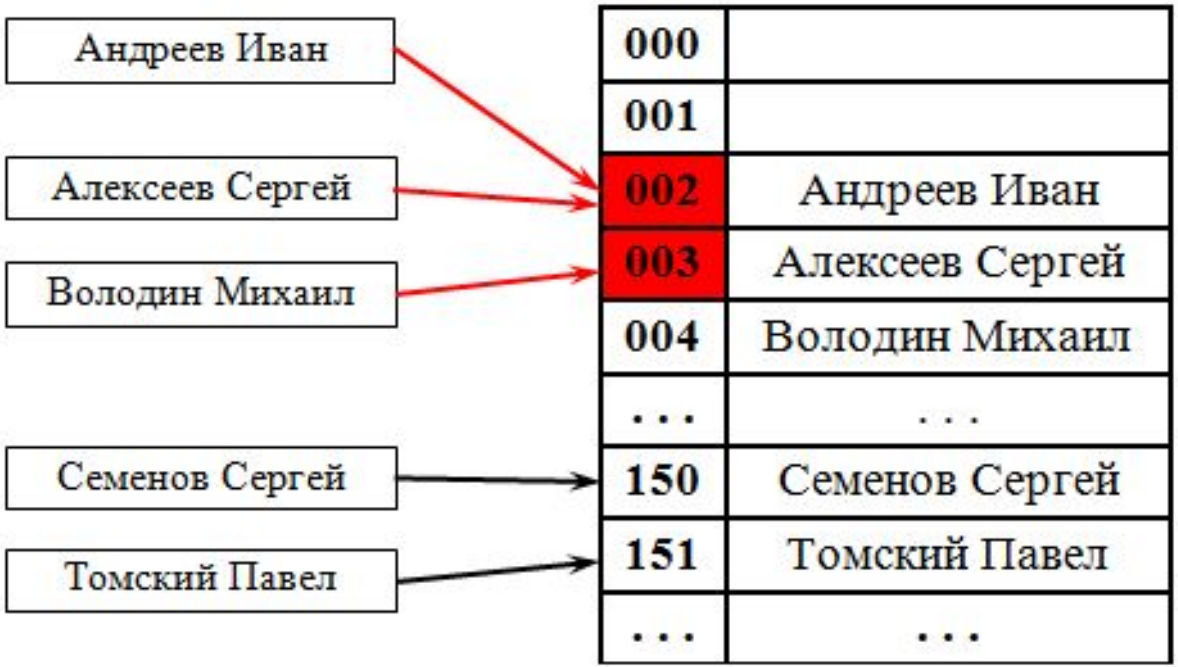
Хэш таблица с прямой адресацией

PROGWARDS
Академия компьютерного мастерства

- Что делать при большом количестве коллизий?
  - увеличение размера таблицы
    - пересчитать все хэш значения
    - перестроить всю таблицу
  - перейти от списков к двоичному дереву

Хэш таблица с открытой адресацией

- Данные хранятся непосредственно в таблице
- Коллизии разрешаются через пробирование

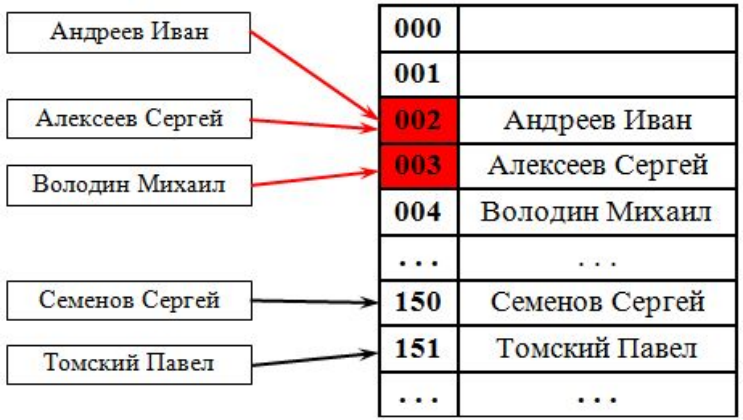


Алгоритмы пробирования

- Линейное пробирование
- Квадратичное пробирование
- Двойное хэширование

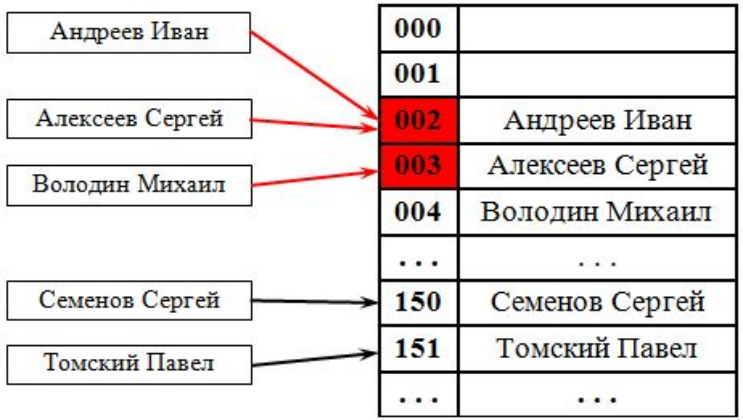
Линейное пробирование

- Поиск свободной ячейки идет последовательным перебором с шагом n
- n должно быть взаимно простым с размером таблицы
- Можно остановиться после просмотра k ячеек, можно просматривать все



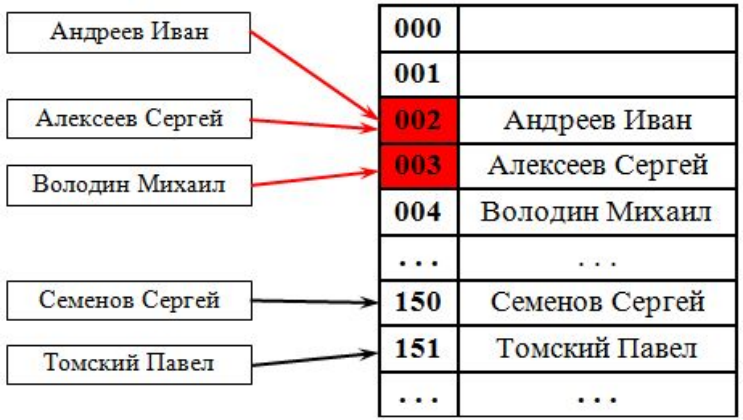
Квадратичное пробирование

- Интервал с каждым шагом увеличивается на какую-то величину
- Если размер таблицы - 2^n, то пример последовательности шагов: 1^2, 2^2, 3^2, ...
- Можно остановиться после просмотра k ячеек, можно просматривать все



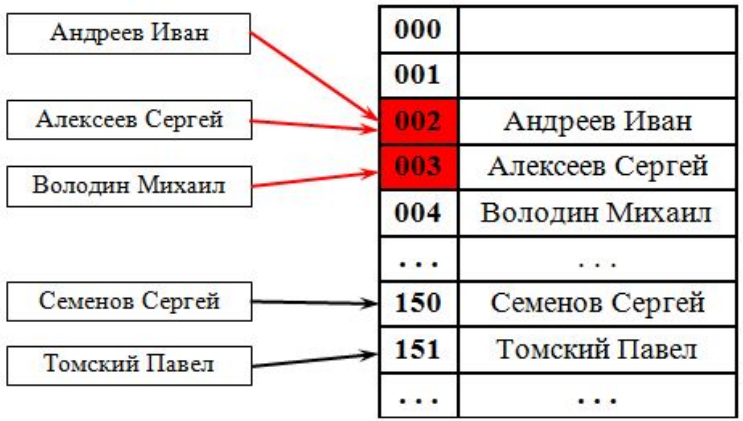
Двойное хеширование

- Поиск свободной ячейки идет последовательным перебором с шагом n
- n вычисляется другой хэш-функцией по тому же ключу
- Может получиться, что не все ячейки будут просмотрены



Проблема удаления

- Присутствует только в таблицах с открытой индексацией
- Не можем помечать ячейку как пустую
- Нужно помечать ячейку как удаленную
- При вставке учитывать эту пометку



Требования к хэш-функции

- Должна быстро вычисляется
- Должна давать минимум коллизий

Виды хэш-функций

- Основанные на делении
- Основанные на умножении
- Хеширование строк переменной длины
- Криптографическое хеширование
- Идеальное хэширование
- Универсальное хэширование

Хэш-функция основанная на делении

PROGWARDS
Академия компьютерного мастерства

- хэш = k % N

  k - ключ
  N - размер таблицы

- рекомендуется брать в качестве N простое число

Хэш-функция основанная на умножении

PROGWARDS
Академия компьютерного мастерства

- хэш = [N*({k*A})]

  k - ключ
  N, A - параметры

- [] - взятие целой части
- {} - взятие дробной части
- () - приоритета операций

Хэш-функция основанная на умножении

PROGWARDS
Академия компьютерного мастерства

- хэш = [N*({k*A})]

  k - ключ
  N, A - параметры

- [] - взятие целой части
- {} - взятие дробной части
- () - приоритета операций

Хэш-функция основанная на умножении

- в качестве A хорошо использовать золотое сечение

$$1/\varphi=(\sqrt{5}-1)/2\approx 0{,}6180339887$$

Хэш-функция основанная на умножении

- в качестве A хорошо использовать золотое сечение

$$1/\varphi=(\sqrt{5}-1)/2\approx 0{,}6180339887$$

Хэш-функция основанная на умножении

```
static int hash(int k) {
   double d = A*k;
   return (int)(N*(d-Math.floor(d)));
}
```

```
int N = 13;
System.out.println(hash(25));
System.out.println(hash(26));
System.out.println(hash(27));
```

вывод на консоль:

```
5
0
8
```

Хэш-функция основанная на умножении

```
static int hash(int k) {
   double d = A*k;
   return (int)(N*(d-Math.floor(d)));
}
```

```
int N = 13;
System.out.println(hash(25));
System.out.println(hash(26));
System.out.println(hash(27));
```

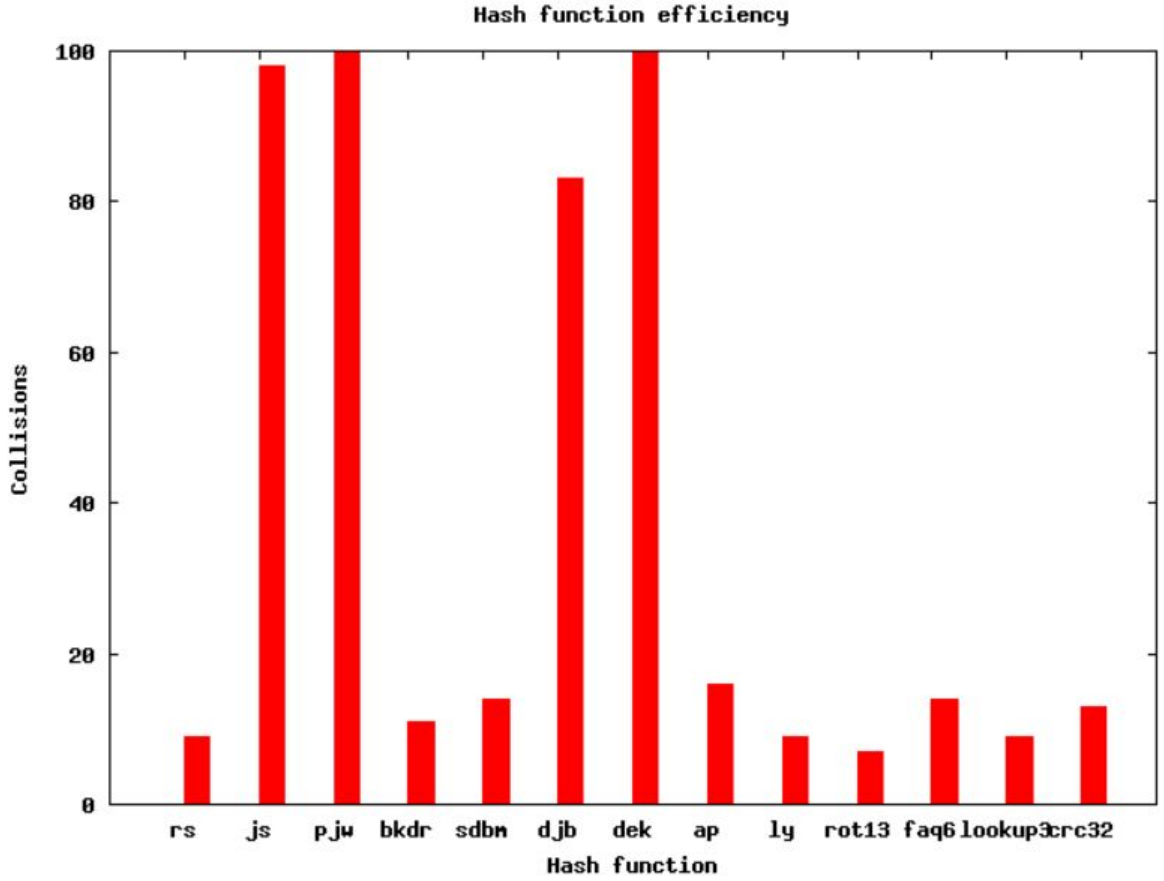
вывод на консоль:

```
5
0
8
```

Хэш-функции для строк

- rs — простая хэш-функция из книги Роберта Седжвика 'Фундаментальные алгоритмы на C'
- js — побитовая хэш-функция от Justin Sobel
- pjw — алгоритм, основанный на работе Peter J. Weinberger
- bkdr — хэш-функция из книги Брайана Кернигана и Денниса Ритчи 'Язык программирования C'
- sdbm — специальный алгоритм, используемый в проекте SDBM
- djb — алгоритм, разработанный профессором Daniel J. Bernstein
- dek — алгоритм, предложенный Дональдом Кнутом в книге 'Искусство программирования'
- ap — алгоритм, разработанный Arash Partow
- faq6 — номер 6 из FAQ Боба Дженкинса
- lookup3 — автор Боб Дженкинс
- ly — предложен Леонидом Юрьевым (конгруэнтный генератор)
- rot13 — простой алгоритм с циклическим сдвигом, от Сергея Вакуленко
- crc32 — стандартная контрольная сумма с полиномом $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

Хэш-функции для строк



Хэш-функции для строк

PROGWARDS
Академия компьютерного мастерства

```
static long RSHash (String str) {
    long b = 378551;
    long a = 63689;
    long hash = 0;
    for (int i = 0; i < str.length(); i++) {
        hash = unsignedInt(hash * a + str.charAt(i));
        a = unsignedInt(a * b);
    }
    return hash;
}
```

Хэш-функции для строк

PROGWARDS
Академия компьютерного мастерства

```
static final long UINT_MAX = 4294967295L;

static long unsignedInt(long num) {
   return num % UINT_MAX;
}
```

Криптографическое хэширование

- Хэш-функция обладает криптостойкостью
- Популярные алгоритмы
  - MD2/4/5/6
  - SHA1/2/3 (224, 256, 384, 512)
- Не используется в хэш-таблицах
- Используется для шифрования паролей, цифровые подписи и т.д.

Идеальное хэширование

- Perfect hash function
- Такое отображение множества ключей S в множество целых чисел N, при котором полностью исключены коллизии
- Строится на фиксированном множестве ключей
- Задача идеального хэширования - построение хэш-функции, дающей такое отображение

Универсальное хэширование

- Universal hashing
- Применяется когда точно не известен характер значений ключей
- Использовать не одну, а семейство хэш-функций, обеспечивающих минимизацию коллизий
- Конкретная хэш-функция выбирается случайным образом по определенной стратегии

Стратегии универсального хэширования

- Менять хэш-функцию раз и n запросов
- Менять хэш-функцию при достижении определенного количества коллизий
- Менять хэш-функцию при каждом перестроении таблицы

Заключение

- Хэш-таблицы позволяют сохранять и получать данные по ключам за константное время O(1)
- Для борьбы с коллизиями используют либо метод цепочек, либо таблицы с открытой адресацией
- Для качественной работы хэш таблиц нужна хорошая хэш-функция
- При известном наборе данных можно использовать идеальное хэширование
- При не известном наборе данных можно использовать универсальное хэширование