EDU
CSDN学院 IT实战派

# 图解数据结构和算法

## 哈希表

讲师：Samuel

# Content

# 哈希表

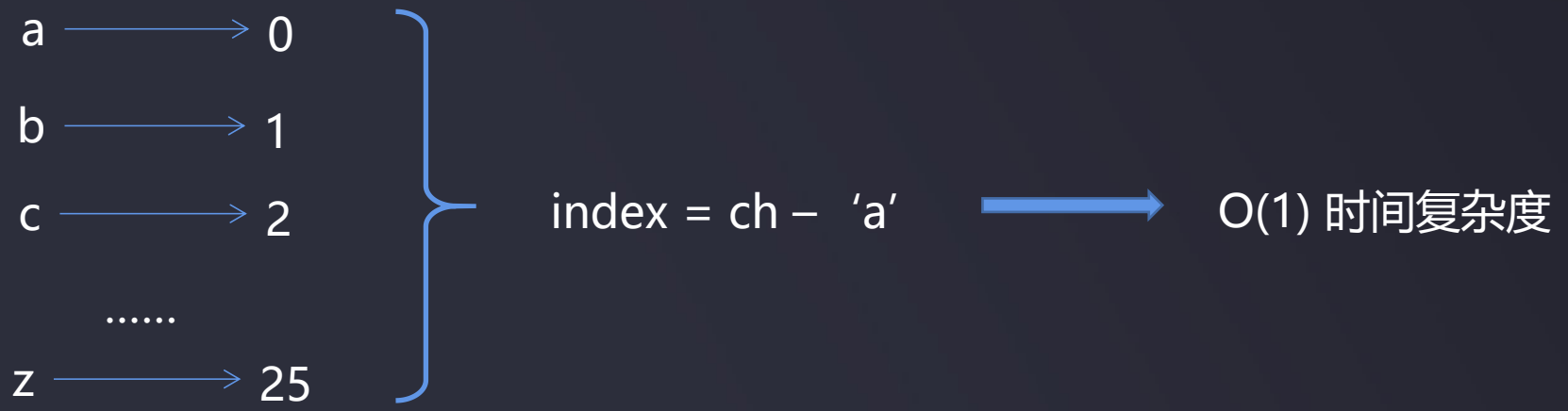
## 1 什么是哈希表

- 哈希表（Hash table），是根据关键码值(Key value)而直接进行访问的数据结构
- 它通过把关键码（key）值映射到表中一个位置来访问记录，以加快查找的速度
- 这个映射函数叫做散列函数，存放记录的数组叫做散列表
- 给定表M，存在函数f(key)，对任意给定的关键字值key
  - 代入函数后若能得到包含该关键字的记录在表中的地址，则称表M为哈希(Hash)表
  - 函数f(key)为哈希(Hash) 函数

# 哈希表

## 2 哈希表举例

- int[] alphabet = new int[26]
- 使用一个长度是26的int数组可以映射出26个字母，这就是一个最简单的哈希表



# 哈希表

## 3 哈希表和哈希函数

- 将键转换成索引的函数就是哈希函数 f(ch)= ch – ‘a’
- 将业务场景中的键转换为索引的过程是哈希表的核心
- 即使再优秀的哈希函数也保证不了一个键对应一个不同的索引，这就是哈希冲突
- 设计哈希函数的原则
  - 一致性：如果a==b，那么hash(a)==hash(b)
  - 高效性： 哈希函数的计算要简单高效
  - 均匀性：尽可能地让哈希值均匀分布
- 哈希表体现了算法设计领域中空间换时间的思想

# 哈希函数

## 1 整型

- 小整数
  - 正整数：直接使用
  - 负整数：偏移成正整数再使用
- 大整数
  - 取部分：eg复杂的学号202021714113
  - 对m取模：m的取值很重要
  - 通常对一个质数取模

| | | | |
|---|---|---|---|
| 10 | %5 | = | 0 |
| 15 | | | 0 |
| 20 | | | 0 |
| 25 | | | 0 |
| 30 | | | 0 |

| | | | |
|---|---|---|---|
| 10 | %7 | = | 3 |
| 15 | | | 6 |
| 20 | | | 2 |
| 25 | | | 4 |
| 30 | | | 1 |

| lwr | upr | % err | prime |
|---|---|---|---|
| $2^{5}$ | $2^{6}$ | 10.416667 | 53 |
| $2^{6}$ | $2^{7}$ | 1.041667 | 97 |
| $2^{7}$ | $2^{8}$ | 0.520833 | 193 |
| $2^{8}$ | $2^{9}$ | 1.302083 | 389 |
| $2^{9}$ | $2^{10}$ | 0.130208 | 769 |
| $2^{10}$ | $2^{11}$ | 0.455729 | 1543 |
| $2^{11}$ | $2^{12}$ | 0.227865 | 3079 |
| $2^{12}$ | $2^{13}$ | 0.113932 | 6151 |
| $2^{13}$ | $2^{14}$ | 0.008138 | 12289 |
| $2^{14}$ | $2^{15}$ | 0.069173 | 24593 |
| $2^{15}$ | $2^{16}$ | 0.010173 | 49157 |
| $2^{16}$ | $2^{17}$ | 0.013224 | 98317 |
| $2^{17}$ | $2^{18}$ | 0.002543 | 196613 |
| $2^{18}$ | $2^{19}$ | 0.006358 | 393241 |
| $2^{19}$ | $2^{20}$ | 0.000127 | 786433 |
| $2^{20}$ | $2^{21}$ | 0.000318 | 1572869 |
| $2^{21}$ | $2^{22}$ | 0.000350 | 3145739 |
| $2^{22}$ | $2^{23}$ | 0.000207 | 6291469 |
| $2^{23}$ | $2^{24}$ | 0.000040 | 12582917 |
| $2^{24}$ | $2^{25}$ | 0.000075 | 25165843 |
| $2^{25}$ | $2^{26}$ | 0.000010 | 50331653 |
| $2^{26}$ | $2^{27}$ | 0.000023 | 100663319 |
| $2^{27}$ | $2^{28}$ | 0.000009 | 201326611 |
| $2^{28}$ | $2^{29}$ | 0.000001 | 402653189 |
| $2^{29}$ | $2^{30}$ | 0.000011 | 805306457 |
| $2^{30}$ | $2^{31}$ | 0.000000 | 1610612741 |

# 哈希函数

## 2 浮点型

- Java中使用float和dubbo来表示浮点数，float占32位、dubbo占64位
- 可以很容易地将浮点型转换成整型
- 然后再遵循整型的哈希函数设计原则来设计哈希函数即可

float f1 = 3.14 → int i1 = f1*100=314 → 小范围正整数，直接取值为哈希函数

# 哈希函数

## 3 字符串

- 字符串类型计算哈希函数核心思想就是将其转化成整型处理
  - 可以直接转化成整型的字符串

    String s1 = “119” -> Integer i1 = Integet.valueOf(s1);
  - 可以通过变换能够转换成整型

    String s1 = “hello”,可以将hello看做是26进制数的表示

| | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|
| Integer(hello) | h*26^4 | e*26^3 | l*26^2 | l*26^1 | o*26^0 | 26进制 |
| Integer(hello) | h*B^4 | e*B^3 | l*B^2 | l*B^1 | o*B^0 | B进制 |

# 哈希函数

## 3 字符串

- 计算出字符串对应的哈希值后就可以对一个质数M取模得到哈希函数
- hash(s1) = (h*B^4+e*B^3+l*B^2+l*B^1+o*B^0)%M
- hash(s1) = (((((h*B)+e)*B+l)*B+l)*B+o)%M
- hash(s1) = (((((h%M) *B+e) %M *B+l) %M *B+l) %M *B+o)%M

```
int hash=0;
String s="hello";
for(int i=0;i<s.length();i++){
    hash = (hash*B+s.charAt(i))%M;
}
```

# 哈希函数

## 4 引用类型

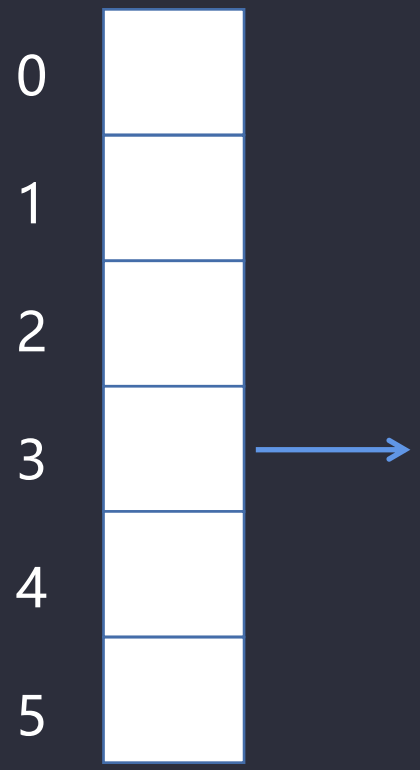
- String是由多个Char组成的一个复杂的引用类型
- 其他引用类型可以直接套用String的哈希函数的公式求得hash值
- hash(stu) = ((((name%M) *B+age) %M *B+sex) %M +address)%M

## 5 Java中的hashCode函数

```
public native int hashCode();
```

# 哈希冲突的处理

## 1 链地址法

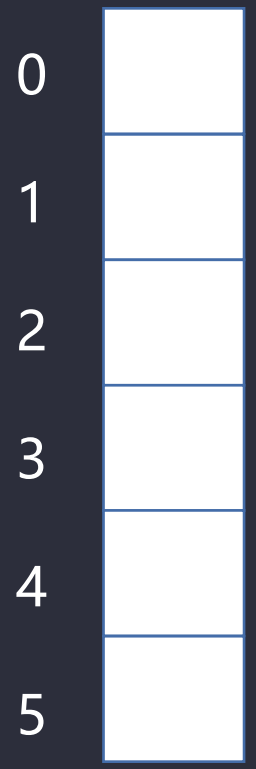


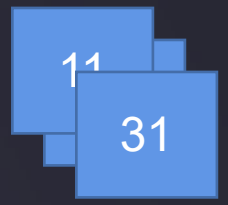


- 先对key使用hash函数计算出hash地址
- hashCode计算出来的地址存在负数
- 可以取正,hashCode(k)&0x7fffffff
- 具有相同的hash地址的key放入到同一个桶中
- HashMap、HashSet底层都是散列表

# 哈希冲突的处理

## 2 开放地址法




hash(k)=key%10

- 线性探测法
- 平方探测
- 再哈希法
- 合理扩容

EDU

CSDN学院 IT实战派

下节课再见，记得关注公众号