9.词典

（c）散列：散列函数

邓俊辉

deng@tsinghua.edu.cn

无法杜绝的冲突

❖ 散列函数 hash(): S ⟼ A，不可能是单射

//词条空间S ~ 可能的词条

|S| = R >> M = |A|

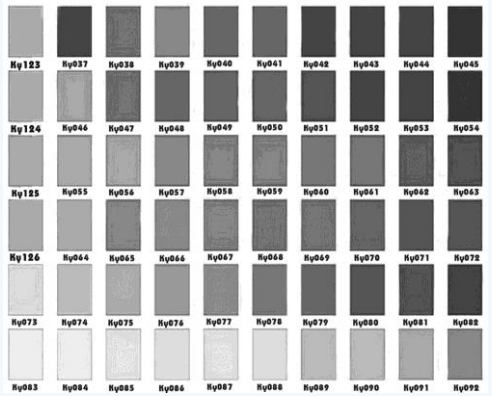
//地址空间A ~ 散列表

两项基本任务

- 近似的单射，往往可行...
- 因此，我们需要...

1） 本节：精心设计散列表及散列函数

以尽可能降低冲突的概率；更需要...



2） 下节：制定可行的预案

以便在发生冲突时，能够尽快予以排解

评价标准与设计原则

❖ 什么样的散列函数hash()更好？

1）确定determinism：　同一关键码总是被映射至同一地址

2）快速efficiency：　expected-$O(1)$

3）满射surjection：　尽可能充分地覆盖整个散列空间

4）均匀uniformity：　关键码映射到散列表各位置的概率尽量接近

　　可有效避免聚集clustering现象

除余法

- hash(key) = key % M

  前例中，为何选M = 90001？

- 若取M = 2^k，其效果相当于

  截取key的最后 k 位（bit），前面的 n - k 位对地址没有影响

  M - 1 = 0 0 0 ... 0 | 1 1 1 … 1

  key % M = key & (M - 1)

  推论：发生冲突 iff 最后k位相同 //发生冲突的概率大

- 若取M ≠ 2^k，缺陷有所改善

- M为素数时，数据对散列表的覆盖最充分，分布最均匀 //蝉的哲学

MAD法

❖ 除余法的缺陷...

1） **不动点**：无论表长M取值如何，总有hash(0) ≡ 0

2） **零阶均匀**：[0, R)的关键码，平均分配至M个桶；但**相邻**关键码的散列地址也必**相邻**

❖ **一阶均匀**：邻近的关键码，散列地址不再邻近 //更高阶的均匀性呢？

❖ MAD = multiply-add-divide

取 M 为素数，a > 0，b > 0，a % M ≠ 0

$$hash(key) = (a \times key + b) \% M$$

❖ 当然，特定场合下，**未必**需要高阶的均匀性

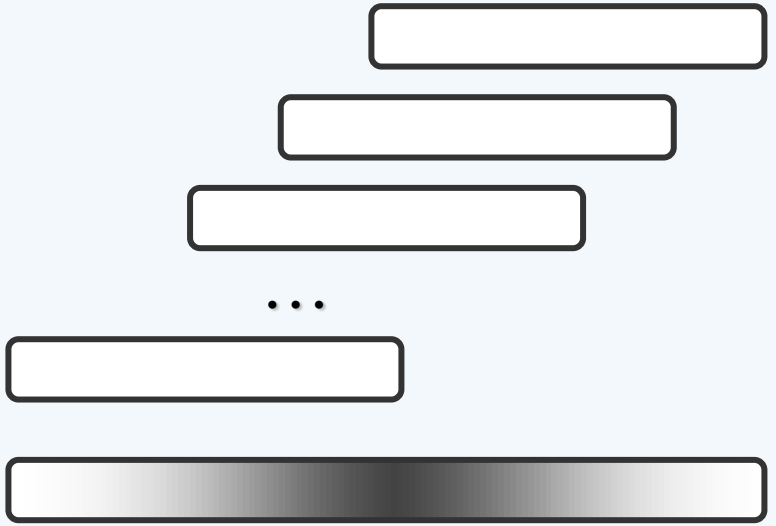
更多散列函数

❖ 数字分析 selecting digits

抽取key中的某几位，构成地址

比如，取十进制表示的奇数位

hash(1 3 5 7 9) = 1 3 5 7 9
  2 4 6 8



❖ 平方取中 mid-square

取key^2的中间若干位，构成地址

hash(123) = 512 //保留$key^2 = 123^2 = 1\boxed{512}9$的中间3位

hash(1234567) = 556 //$1234567^2 = 15241\boxed{556}77489$

更多散列函数

- 折叠法folding：将key分割成等宽的若干段，取其总和作为地址

  hash($_{123}456^{789}$) = 1368 //123 + 456 + 789，自左向右

  hash($_{123}456^{789}$) = 1566 //123 + 654 + 789，往复折返

- 位异或法XOR：将key分割成等宽的二进制段，经异或运算得到地址

  hash($_{110}011^{011}{}_{b}$) = 110_b //110 ^ 011 ^ 011，自左向右

  hash($_{110}011^{011}{}_{b}$) = 011_b //110 ^ 110 ^ 011，往复折返

- ...

- 总之，越是随机，越是没有规律，越好

（伪）随机数法

❖ （伪）随机数发生器

循环：rand(x + 1) = [a × rand(x)] % M //M素数，a % M ≠ 0

a = 7^5 = 16,807 = 10000110100111_b

M = 2^{31} - 1 = 2,147,483,647 = 01111111 11111111 11111111 11111111_b

❖ （伪）随机数法

径取：hash(key) = rand(key) = [rand(0) × a^{key}] % M

种子：rand(0) = ?

❖ 把难题推给伪随机数发生器，但是...

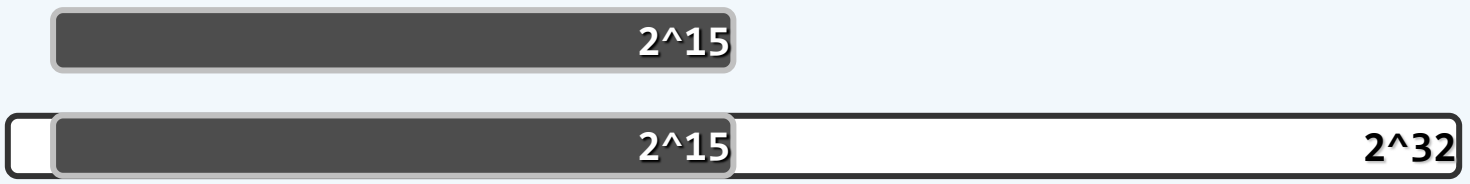
❖ （伪）随机数发生器的实现，因具体平台、不同历史版本而异

创建的散列表可移植性差——故需慎用此法！

（伪）随机数法

❖ unsigned long int next = 1; //The C Programming Language (2nd edn), p46

```
void srand(unsigned int seed) {  next = seed;  }
int rand(void) { //1103515245  =  3^5 * 5 * 7 * 129749
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}
```



❖ 另类随机数（串）算法

```
int rand() { int uninitialized; return uninitialied; }
char* rand( t_size n ) { return ( char* ) malloc( n ); }
```

❖ 以上方法，可否用于散列？

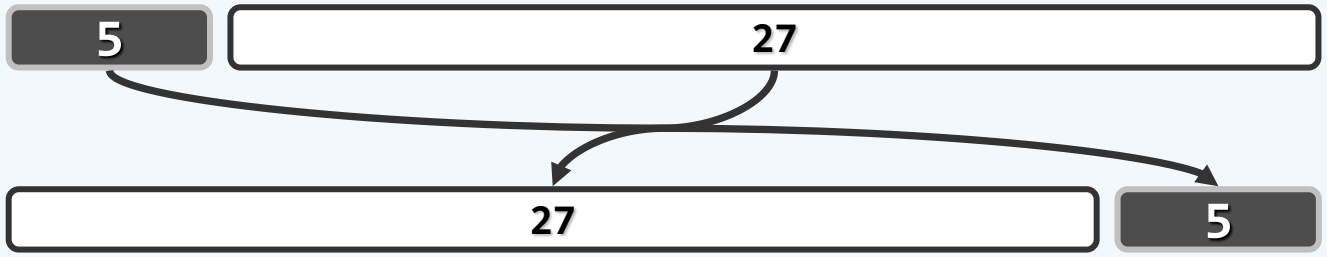
多项式法

❖ hash(s = $x_0\ x_1\ \ldots\ x_{n-1}$) = $x_0 \cdot a^{n-1} + x_1 \cdot a^{n-2} + \ldots + x_{n-2} \cdot a^1 + x_{n-1}$

= $(\ \ldots\ (\ (\ (\ x_0 * a\ + x_1\ )\ * a\ + x_2\ )\ * a\ + \ldots\ x_{n-2}\ )\ * a\ + x_{n-1}$

```
❖ static size_t hashCode( char s[] ) { //近似多项式，但更快捷
    int h = 0;
    for ( size_t n = strlen(s), i = 0; i < n; i++ )
       { h = (h << 5) | (h >> 27); h += ( int ) s[i]; }
    return ( size_t ) h;
  } //有必要如此复杂吗？
```

❖ 能否使用更简单的散列，比如...



多项式法

- 字符分别映射为数字：f(c) = CODE(UPPER(c)) - 64　　//['A', 'Z'] ~ [1, 26]

  再简单相加：hash(s) = $\sum_{c \in s} f(c)$

  hash ~ 8 + 1 + 19 + 8 = 36

- 字符相对次序信息丢失，将引发大量冲突

  I am Lord Voldemort

  Tom Marvolo Riddle

- 即便字符不同、数目不等...

  He's Harry Potter

- Key to improving your programming skills

  Learning Tsinghua Data Structures & Algorithms

Java:hashCode()

- hashCode()方法

  适用于Java中的所有对象

  将任意类的对象转换为（32位int型）整数

  对于非整型的key，先转换为整数（散列码），然后再做散列

- hashCode()：效果如何？效率如何？是如何实现的？

- object.hashCode() = object在内存中的地址

- 问题：相关的对象地址也相近，冲突概率高 //更糟糕的是...

- 散列码与对象的内容无关

  比如，完全相等的两个字符串对象，散列码居然不同

- 有何替代方案？