数据结构与算法期末复习
Homework

尹超

中国科学院大学，北京 100049

Carter Yin

University of Chinese Academy of Sciences, Beijing 100049, China

2024.8 – 2025.1

序言

本文为笔者数据结构与算法的期末复习笔记。
望老师批评指正。

目录

题目 第八章词典

143

> 散列函数是 h(x) = x % 20，则关键码 25, 85, 15, 20 中会发生冲突的是:
> A. 25 和 85
> B. 25 和 15
> C. 15 和 85
> D. 20 和 15

Solution 1. 正确答案是 *A*。

详细分析:

散列冲突（*Hash Collision*）是指两个或多个不同的关键码（*key*）通过同一个散列函数（*hash function*）计算后得到了相同的散列地址（*hash value*）。

给定的散列函数是 *'h(x) = x*

- ***h(25)*** *= 25 % 20 =* ***5***
- ***h(85)*** $= 85\%20 =$ ***5*** （因为 $85 = 4 \times 20 + 5$）
- ***h(15)*** *= 15 % 20 =* ***15***
- ***h(20)*** *= 20 % 20 =* ***0***

通过计算可以发现，关键码 ***25*** 和 ***85*** 都得到了相同的散列地址 ***5***。因此，它们会发生冲突。

144

> 在散列中，关键码个数超过实际使用的空间时，有没有可能不发生冲突:
> A. 可能
> B. 不可能

Solution 2. 正确答案是 *B*。

详细分析:

这个问题可以用 ** 鸽巢原理（*Pigeonhole Principle*）** 来解释。

- **鸽巢原理:** 如果有 $n+1$ 只鸽子要飞进 n 个鸽巢，那么至少有一个鸽巢里会有两只或更多的鸽子。

在散列的上下文中:

- “关键码”相当于“鸽子”。
- “实际使用的空间”(即散列表的槽位/桶）相当于“鸽巢”。

当“关键码个数”(鸽子数）超过“实际使用的空间”(鸽巢数）时，根据鸽巢原理，必然会有至少两个不同的关键码被散列函数映射到同一个空间位置。

这种情况就是 ** 散列冲突 **。

因此，当关键码的数量超过散列表的大小时，冲突是 ** 不可避免 ** 的，绝对会发生。

145

> S 为所有可能词条的空间，A 为所有可用地址的空间（|A| < |S|），h 是散列函数，则:
> A. 从 S 映射到 A，一定是满射
> B. 从 S 映射到 A，不可能是单射
> C. 从 A 映射到 S，不可能是满射
> D. 从 A 映射到 S，一定是单射

Solution 3. 正确答案是 B。

详细分析:

我们来分析一下函数映射的性质:

- **单射 *(Injection):*** 如果集合 S 中的每一个不同的元素，通过函数 h 映射到集合 A 中的元素也都是不同的，那么这个映射就是单射。即，如果 $s_1 \neq s_2$，则 $h(s_1) \neq h(s_2)$。
- **满射 *(Surjection):*** 如果集合 A 中的每一个元素，都至少有一个 S 中的元素与之对应，那么这个映射就是满射。

题目给出的条件是:

- S 是定义域（所有可能的词条）。
- A 是陪域（所有可用的地址）。
- h 是从 S 到 A 的映射 ($h : S \to A$)。
- 关键条件是 $|A| < |S|$，即地址空间的大小小于词条空间的大小。

现在我们来评估每个选项:

- ***A.* 从 S 映射到 A，一定是满射:** 这个不一定。一个设计糟糕的散列函数可能将所有词条都映射到 A 中的同一个地址，这样 A 中其他地址就没有被用到，因此不是满射。
- ***B.* 从 S 映射到 A，不可能是单射:** 这个是正确的。根据鸽巢原理，当"鸽子"(S 中的元素）的数量大于"鸽巢"(A 中的元素）的数量时，必然至少有两个鸽子要共享同一个鸽巢。同理，当 $|S| > |A|$ 时，必然存在至少两个不同的词条 s_1, s_2 被映射到同一个地址，即 $h(s_1) = h(s_2)$。这违反了单射的定义。因此，这个映射不可能是单射。
- ***C.* 从 A 映射到 S，不可能是满射:** 这句话本身是正确的（因为 $|A| < |S|$），但它描述的是一个从 A 到 S 的映射，而散列函数 h 是从 S 到 A 的映射。所以这个选项与题目描述的散列函数 h 无关。
- ***D.* 从 A 映射到 S，一定是单射:** 这句话是错误的。我们可以定义一个从 A 到 S 的非单射函数。同样，这个选项也与散列函数 h 无关。

因此，唯一正确描述散列函数 h 性质的选项是 B。

146

> 考虑 key 的集合 S = {0, 8, 16, 24, 32, 40, 48, 56, 64}
> 用除余法构造的散列函数
> h1(key) = key % 12
> h2(key) = key % 11
> h1 将 S 映射到的值域有几个元素? ___________
> h2 将 S 映射到的值域有几个元素? ___________

Solution 4. **详细分析:**

对于 h1(key) = key % 12

我们对集合 S 中的每个元素计算其散列值:

- *h1(0) = 0 % 12 =* ***0***
- *h1(8) = 8 % 12 =* ***8***
- *h1(16) = 16 % 12 =* ***4***
- *h1(24) = 24 % 12 =* ***0***
- *h1(32) = 32 % 12 =* ***8***
- *h1(40) = 40 % 12 =* ***4***
- *h1(48) = 48 % 12 =* ***0***
- *h1(56) = 56 % 12 =* ***8***
- *h1(64) = 64 % 12 =* ***4***

h1 映射得到的值的集合（值域）为 *{0, 4, 8}*。该集合中有 ***3*** 个元素。

原因分析: 所有的 *key* 都是 *8* 的倍数，而散列表长度为 *12*。*'gcd(8, 12) = 4'*，这导致了严重的聚集，散列值只会是 *4* 的倍数（*0, 4, 8*）。

对于 h2(key) = key % 11

我们对集合 S 中的每个元素计算其散列值:

- *h2(0) = 0 % 11 =* ***0***
- *h2(8) = 8 % 11 =* ***8***
- *h2(16) = 16 % 11 =* ***5***
- *h2(24) = 24 % 11 =* ***2***
- *h2(32) = 32 % 11 =* ***10***
- *h2(40) = 40 % 11 =* ***7***
- *h2(48) = 48 % 11 =* ***4***
- *h2(56) = 56 % 11 =* ***1***
- *h2(64) = 64 % 11 =* ***9***

h2 映射得到的值的集合（值域）为 *{0, 1, 2, 4, 5, 7, 8, 9, 10}*。该集合中有 ***9*** 个元素。

原因分析: 散列表长度为 *11*，是一个素数，且与 *key* 的公因子 *8* 互质（*'gcd(8, 11) = 1'*）。这使得散列值分布得非常均匀，对于 S 中的 *9* 个不同输入，产生了 *9* 个不同的输出。

---

h1 将 S 映射到的值域有几个元素? ***3***

h2 将 S 映射到的值域有几个元素? ***9***

147

> 关于排解冲突的方法，以下说法正确的是: A. 用独立链法排解冲突，所有词条的实际存放位置均在桶数组内部 B. 用开放定址排解冲突，词条的实际存放位置不一定是对应的散列函数值 C. 用开放定址排解冲突，词条被存放在列表中 D. 只要散列函数设计得当，不一定需要排解冲突的策略

Solution 5. 正确答案是 *B*。

详细分析：

- ***A.* 用独立链法排解冲突，所有词条的实际存放位置均在桶数组内部** 这个说法是**错误**的。独立链法（*Separate Chaining*）是在桶数组的每个槽位上维护一个次级数据结构（通常是链表）。桶数组本身只存放指向链表头部的指针或引用。发生冲突的词条被添加到这个链表中，因此它们实际存放在桶数组**外部**的动态分配的内存中。
- ***B.* 用开放定址排解冲突，词条的实际存放位置不一定是对应的散列函数值** 这个说法是**正确**的。开放定址（*Open Addressing*）的核心思想是：当一个词条的散列地址 *'h(key)'* 已经被占用时，就去探测（*probe*）散列表中的其他位置，直到找到一个空槽位来存放该词条。因此，除了第一个没有发生冲突的词条外，其他词条的实际存放位置很可能不是其初始散列函数计算出的地址。
- ***C.* 用开放定址排解冲突，词条被存放在列表中** 这个说法是**错误**的。这是对独立链法的描述。开放定址法的所有词条都直接存放在散列表（即桶数组）的内部，不使用外部的链表。
- ***D.* 只要散列函数设计得当，不一定需要排解冲突的策略** 这个说法是**错误**的。根据鸽巢原理，只要可能输入的关键码总数大于散列表的槽位数，冲突就是理论上不可避免的。一个好的散列函数只能尽可能地**减少**冲突的概率，使其分布均匀，但**不能完全消除**冲突。因此，任何一个通用的散列表实现都必须有排解冲突的策略。

148

> 规模为 11 的桶数组当前状态为 A = { *, *, *, *, *, 0, 15, 26, *, 5, 9}，其中 * 表示空桶
> 散列函数为 h(key) = (3 * key + 5) % 11
> 用开放定址 + 线性试探排解冲突
> 插入词条 4, 它的实际存放位置是
> A. A[4]
> B. A[6]
> C. A[7]
> D. A[8]

Solution 6. 正确答案是 *D*。

详细分析：

(1) **计算初始散列地址：** 首先，我们用散列函数计算词条 *4* 的初始散列地址。

- *'h(4) = (3 * 4 + 5)*
- *'h(4) = (12 + 5)*
- *'h(4) = 17*
- *'h(4) = 6'*

所以，词条 *4* 的理想存放位置是 *'A[6]'*。

(2) **检查冲突：** 我们查看桶数组 *'A'* 在索引 *6* 的位置。*'A[6]'* 的值为 *15*，不是空的。因此，发生了冲突。

(3) **应用线性试探：** 由于发生了冲突，我们需要使用线性试探（*Linear Probing*）来寻找下一个可用的空桶。线性试探就是依次检查下一个位置。

- **第一次试探：** 检查 *'A[(6 + 1)*
- **第二次试探：** 检查 *'A[(6 + 2)*

(4) **确定存放位置：** 因为 *'A[8]'* 是沿着试探路径找到的第一个空桶，所以词条 *4* 将被存放在 *'A[8]'*。

149

> 规模为 11 的桶数组当前状态为 A = { *, *, *, *, *, 0, 15, 26, *, 5, 9}，其中 * 表示空桶
> 散列函数为 h(key) = (3 * key + 5) % 11
> 用开放定址 + 平方试探排解冲突
> 插入词条 4, 它的实际存放位置是
> A. A[4]
> B. A[6]
> C. A[7]
> D. A[8]

Solution 7. 正确答案是 A。

详细分析:

(1) **计算初始散列地址:** 首先，我们用散列函数计算词条 *4* 的初始散列地址。
- *‘h(4) = (3 * 4 + 5)*
- *‘h(4) = (12 + 5)*
- *‘h(4) = 17*
- *‘h(4) = 6‘*

所以，词条 *4* 的理想存放位置是 *‘A[6]‘*。

(2) **检查冲突:** 我们查看桶数组 *‘A‘* 在索引 *6* 的位置。*‘A[6]‘* 的值为 *15*，不是空的。因此，发生了冲突。

(3) **应用平方试探:** 由于发生了冲突，我们需要使用平方试探（*Quadratic Probing*）来寻找下一个可用的空桶。平方试探的地址序列是 $H_i = (h(key) + i^2) \pmod m$，其中 $i = 1, 2, 3, \ldots$。
- **第一次试探 *(i=1)*:** 检查 $A[(6 + 1^2)\%11]$，即 $A[7]$。$A[7]$ 的值为 *26*，仍然被占用。
- **第二次试探 *(i=2)*:** 检查 $A[(6 + 2^2) \bmod 11]$，即 $A[(6 + 4) \bmod 11]$，即 $A[10]$。$A[10]$ 的值为 *9*，仍然被占用。
- **第三次试探 *(i=3)*:** 检查 $A[(6+3^2) \bmod 11]$，即 $A[(6+9) \bmod 11]$，即 $A[15 \bmod 11]$，即 $A[4]$。$A[4]$ 的值是 $*$，表示这是一个空桶。

(4) **确定存放位置:** 因为 *‘A[4]‘* 是沿着试探路径找到的第一个空桶，所以词条 *4* 将被存放在 *‘A[4]‘*。

150

> 散列表的规模是素数，用开放定址 + 平方试探法排解冲突，若要保证新的词条能够顺利插入，散列表的装填因子不能超过（请填十进制小数）__________

Solution 8. **详细分析:**

(1) **平方试探的性质:** 平方试探（*Quadratic Probing*）的一个重要特性是，它不能保证在散列表未满时总能找到一个空槽位。它的探测序列可能会在一个子集内循环，从而错过其他可用的空槽。

(2) **保证插入成功的条件:** 有一个重要的定理指出:

> 如果散列表的大小 m 是一个素数，并且装填因子 α *(*即 $\alpha = \frac{\text{已存词条数}}{\text{散列表大小}}$*)* 不超过 *0.5*，那么使用平方试探法（探测序列为 $h_i = (h(key) + i^2) \pmod m$）总能为新词条找到一个空槽位。

(3) **原因简述**：该定理的证明基于以下事实：当 m 为素数且 $\alpha \leq 0.5$ 时，前 $\lceil m/2 \rceil$ 次探测（包括初始位置）所访问的地址都是互不相同的。因为散列表至少有一半是空的，所以在这前 $\lceil m/2 \rceil$ 次探测中，必然会遇到一个空槽位。

(4) **结论**：如果装填因子超过 *0.5*，就无法保证一定能找到空位。因此，为了确保新的词条总能顺利插入，装填因子不能超过 *0.5*。

散列表的规模是素数，用开放定址 + 平方试探法排解冲突，若要保证新的词条能够顺利插入，散列表的装填因子不能超过（请填十进制小数）_____***0.5***_____