第9章

词典

[9-1] 阅读教材代码 9.7（253 页）、代码 9.8（255 页）和代码 9.11（258 页）。
试验证：本章所实现的跳转表结构，可保证雷同的词条在内部按插入次序排列，同时对外先进先出。

【解答】

由代码9.8可见，算法Skiplist::put(k, v)总是找到不大于k的最后一个节点p，并紧邻于p所属塔的右侧为(k, v)创建一座新塔。因此，雷同词条在该结构内部的存储次序，就是完全按照插入次序的“先左后右”。

由代码9.7可见，算法Skiplist::skipSearch()是自左向右查找第一个命中的词条。因此若有多个雷同词条，如代码9.11所示的Skiplist::remove()算法所删除的，必然是最早插入跳转表的词条。

可见，该结构的操作接口的确符合“先进先出”的语义要求。

[9-2] 本章所实现的跳转表结构中，每个词条都在所属的塔内同时保留了多个副本。尽管这样可以简化代码描述，但毕竟浪费了大量的空间，在词条本身较为复杂时由其如此。
试在本章相关代码的基础上就此做一改进，使得每座塔仅需保留一份对应的词条。

【解答】

比如，只需将所有词条组织为一个独立的横向列表，则各词条所对应的纵向列表（塔）即可不必重复保留词条的副本。纵向列表中的每个节点，只需通过引用指向横向列表中对应的词条。如此，一旦查找终止于某纵向列表，即可直接通过引用找到对应的词条。

请读者按照以上介绍和提示，独立完成编码和调试任务。

[9-3] W. Pugh 曾经通过实验统计，将 skiplist（A）与非递归版 AVL 树（C）、伸展树（B）、递归版 (2, 3)-树（D）等数据结构做过对比，并发现了以下规律：

(a) 就 search()/get()接口的效率而言，B 最优
(b) 就 insert()/put()接口的效率而言，A 最优，C 优于 D
(c) 就 remove()接口的效率而言，A 最优

试通过实验核对他的结论，并结合本书对这些结构的讲解，对以上规律作出直观的解释。

【解答】

请读者独立完成测试任务，根据统计结果作出结论，并结合自己的理解作出解释。

[9-4] 为便于客户记忆，许多商家都将其产品销售咨询电话号码与公司或产品的名称直接关联。其中最流行的一种做法可以理解为，在电话键盘的拨号键与数字之间建立一个散列映射：

```
{'A', 'B', 'C'} → 2   {'D', 'E', 'F'} → 3   {'G', 'H', 'I'} → 4
{'J', 'K', 'L'} → 5   {'M', 'N', 'O'} → 6   {'P', 'Q', 'R', 'S'} → 7
{'T', 'U', 'V'} → 8   {'W', 'X', 'Y', 'Z'} → 9
```

比如，IBM 公司的销售电话：

```
+1 (800) 426-7253
```

即对应于字符串

```
"IBM-SALE"
```

又如，Dell 公司的销售电话

```
+1 (888) xxx-3355
```

则对应于字符串

```
"DELL"
```

如此，客户只需记住对应的有意义字符串，而不再是枯燥乏味的数字。

请留意观察身边的这类现象，找出更多这样的实例。

【解答】

再如，联想在北美的销售电话

```
+1 (855) 253-6686
```

即对应于字符串

```
"LENOVO"
```

更多实例的发现，请读者通过观察独立完成。

[9-5] 实际上早在上世纪 70 年代，Bell 实验室就已采用上题中的散列映射法，根据员工的姓名分配办公电话，且可轻松地将发生冲突的概率降至 0.2%以下。

a） 这一方法是否适用于中文（拼音）姓名？

【解答】

与英文相比，拼音中各字母出现频度的分布有很大差异，而且相邻字母组合的情况也很不一样，再加上大量同音字等因素，照搬原方法未必能够适合中文姓名。

b） 试以你所在班同学的姓名（拼音）为样本做一实验，并分析你的实验结果。

【解答】

请读者独立完成测试任务，并根据统计结果作出分析和判断。

[9-6] 假定散列表长度为 M，采用模余法。若从空开始将间隔为 T 的 M 个关键码插入其中。试证明，若 g = gcd(M, T)为 M 和 T 的最大公约数，则

a） 每个关键码均大约与 g 个关键码冲突；

【解答】

这一组关键码依次构成一个等差数列，其公差为T。不失一般性，设它们分别是：

$\{ 0, T, 2T, 3T, \ldots, (M - 1)T \}$

按照模余法，任何一对关键码相互冲突，当且仅当它们关于散列表长M，属于同一同余类。这里，既然g是M和T的最大公约数，故相对于M而言，这些关键码分别来自M/g个同余类，且每一类各有彼此冲突的g个关键码。例如，其中0所属的同余类为：

$\{ 0, TM/g, 2TM/g, 3TM/g, \ldots, (g-1)TM/g \}$

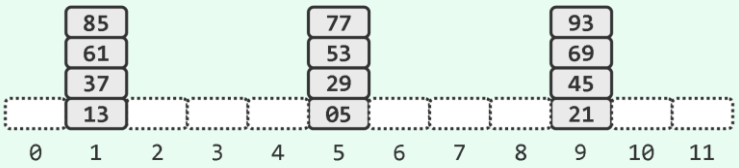
例如，若将首项为5、公差为8的等差数列：

$\{ 5, 13, 21, 29, 37, 45, 53, 61, 69, 77, 85, 93 \}$

作为词条插入长度M = 12的散列表，则如图x9.1所示，这些词条将分成：

$M/g = 12/\gcd(12, 8) = 12/4 = 3$

个冲突的组，各组均含4个词条。



图x9.1 表长与公差有非平凡公因子时，会出现大量的冲突

不难验证，只要是这12个词条，则无论其插入的次序如何，冲突的情况都与此大同小异。

另外，采用MAD法的冲突情况也颇为类似。就其效果而言，在MAD法所用散列函数：

$(a \times key + b) \bmod M$

中，b即相当于以上算术级数的首项，a即相当于公差T。比如，就上例而言，有：

$a = 8$ 和 $b = 5$

由此也可从一个侧面理解，为何M和a的取值通常都必须二者互素——如此便有：

$M / \gcd(M, a) = M/1 = M$

从而最大程度地保证散列的随机性和均匀性。

b） 如不采取排解冲突的措施，散列表的利用率将约为 1/g。

【解答】

根据以上分析，散列表中的M个桶与M的M个同余类一一对应。既然此时的关键码只可能来自其中的M/g个同余类，故必有M - M/g个桶闲置，空间利用率不超过：

$$\frac{M/g}{M} = 1/g$$

[9-7] 我们已经看到，散列表长度 M 是影响散列效果的重要因素之一。
为保证散列映射的均匀性和随机性，M 的取值，应能避免后续查询和修改操作可能的非随机性。
试说明：就以上意义而言，表长不宜取作 $M = 2^k$ ($k \geq 2$)。

【解答】

若取$M = 2^k$，则对任何词条key都有：

$$key \ \% \ M \ = \ key \ \& \ (M - 1) \ = \ key \ \& \ 00...0\boxed{11...1}$$

其中，“%”和“&”分别是算术取模运算和逻辑位与运算，$00...0\boxed{11...1}$中共含k个“1”。

于是，此时采用模余法的效果，等同于从key的二进制展开式中截取末尾的k个比特。也就是说，词条key更高的其余比特位对散列的位置没有任何影响，从而在很大程度上降低了散列的随机性和均匀性。

[9-8] 试证明，23 人中存在生日巧合的概率大于 50%。

【解答】

不妨将每年的365天视作长度为365的散列表，将每个人视作一个待插入的词条。于是，考查已累计插入n个词条时的情况，若将此时至少有一对词条发生冲突的概率记作P(n)，则应有：

$$1 - P(n) \ = \ 365/365 \times 364/365 \times 363/365 \times 362/365 \times ... \times (366 - n)/365$$

$$P(n) \ = \ 1 - 365/365 \times 364/365 \times 363/365 \times 362/365 \times ... \times (366 - n)/365$$

表x9.1 n个人中存在生日巧合的概率

n	1	2	3	4	5	6	7	8	9	10
P(n) (%)	0.00	0.27	0.82	1.64	2.71	4.05	5.62	7.43	9.46	11.69

n	11	12	13	14	15	16	17	18	19	20
P(n) (%)	14.11	16.70	19.44	22.31	25.29	28.36	31.50	34.69	37.91	41.14

n	21	22	23	24	25	26	27	28	29	30
P(n) (%)	44.37	47.57	50.73	53.83	56.87	59.82	62.69	65.45	68.10	70.63

对于不同的n，P(n)的取值如表x9.1所示。可见，当$n \geq 23$后，即有$P(n) \geq 50\%$。

[9-9] 在本章示例代码的基础上进行扩充，实现线性试探以外的其它冲突排解策略。

【解答】

请读者根据教材中对相关策略及方法的介绍，独立完成编码和调试任务。

[9-10] 若允许关键码雷同的词条并存，本章实现散列表结构的示例代码应该如何修改？

【解答】

在操作语义方面，查找接口、删除接口的返回值应调整为“与目标词条相等的任一词条”。在逻辑控制方面，插入接口即便已经发现雷同词条，也要重复插入，因此总是以成功返回。

请读者根据以上提示，针对不同的散列及排解冲突策略，独立完成改进任务。

[9-11] 创建散列表结构时，通常首先需要初始化所有的桶单元。尽管如 265 页代码 9.14 所示，这可以借助系统调用 memset()实现，但所需的时间将线性正比于散列表的长度。

a） 试设计一种方法将初始化时间减至 $O(1)$，而且此后在查找或插入词条时，仅需 $O(1)$时间即可判定任何桶是否处于初始状态；（提示：参考文献[4][9]）

【解答】

借助习题[2-34]中实现的Bitmap类及其技巧。

b） 你的方法需要额外使用多少空间？是否会因此提高散列表的整体渐进空间复杂度？

【解答】

这里引入的Bitmap结构，与散列表等长，故整体的渐进空间复杂度保持不变。

c） 继续扩展你的方法，使之支持删除操作——桶被清空之后，用 $O(1)$时间将其恢复为初始状态。

【解答】

同样地，借助习题[2-34]中实现的Bitmap类及其技巧。

[9-12] a） 在平方试探法、（伪）随机试探法等方法中，查找链如何构成？

【解答】

与线性试探法不同，此时构成查找链的各桶未必彼此相邻；但同样地，不同的查找链仍可能相互有所重叠。

b） 如何调整和推广懒惰删除法，使之可以应用于这些闭散列策略？

【解答】

仿照线性试探法，只不过需要按照新的试探策略遍历查找链。

[9-13] 在实现平方试探法时，可否只使用加法而避免乘法（平方）运算？如果可以，试给出具体方法。

【解答】

事实上，若散列表长为M，则对于任意非负整数k，都有：

$$(k + 1)^2 \equiv k^2 + (k + k + 1) \pmod{M}$$

只要注意到这一规律，在前一桶地址（k^2 % M）的基础上，只需再做三次加法运算，即可得到下一桶地址（$(k + 1)^2$ % M）。

[9-14] 考查单向平方试探法，设散列表长度取作素数 M > 2。试证明：

a） 任一关键码所对应的查找链中，前$\lceil M/2 \rceil = (M + 1)/2$ 个桶必然互异；

（提示：只需证明，{ 0^2, 1^2, 2^2, ..., $\lfloor M/2 \rfloor^2$ }关于 M 分别属于不同的同余类）

【解答】

反证。假设存在$0 \le a < b < \lceil M/2 \rceil$，使得查找链上的第a个位置与第b个位置冲突，于是$a^2$和$b^2$必然同属于关于M的某一同余类，亦即：

$$a^2 \equiv b^2 \pmod{M}$$

于是便有：

$$a^2 - b^2 = (a + b)\cdot(a - b) \equiv 0 \pmod{M}$$

然而，无论是(a + b)还是(a - b)，绝对值都严格小于M，故均不可能被M整除——这与M是素数的条件矛盾。

b） 在装填因子尚未增至50%之前，插入操作必然成功（而不致因无法抵达空桶而失败）；

【解答】

由上可知，查找链的前⌈M/2⌉项关于M，必然属于不同的同余类，也因此互不冲突。在装填因子尚不足50%时，这⌈M/2⌉项中至少有一个是空余的，因此不可能发生无法抵达空桶的情况。

c） 在装填因子超过50%之后，只要适当调整各桶的位置，下一插入操作必然因无法抵达空桶而失败。（提示：只需证明，$\{ 0^2, 1^2, 2^2, \ldots \}$关于M的同余类恰好只有⌈M/2⌉个）

【解答】

任取：

$$\lceil M/2 \rceil \le c < M - 1$$

并考查查找链上的第c项。

可以证明，总是存在$0 \le d < \lceil M/2 \rceil$，且查找链上的第d项与该第c项冲突。

实际上，只要令：

$$d = M - c \neq c$$

则有：

$$c^2 - d^2 = (c + d)\cdot(c - d) = M\cdot(c - d) \equiv 0 \pmod{M}$$

于是c^2和d^2关于M同属一个同余类，作为散列地址相互冲突。

[9-15] a） 试举例说明，散列表长度M为合数时，即便装填因子低于50%，平方试探仍有可能无法终止；

【解答】

考查长度为M = 12的散列表。

不难验证，$\{ 0^2, 1^2, 2^2, 3^2, 4^2, \ldots \}$关于M模余数只有{ 0, 1, 4, 9 }四种可能。于是，即便只有这四个位置非空，也会因为查找链的重合循环，导致新的关键码0无法插入。而实际上，此时的装填因子仅为：

$$\lambda = 4/12 < 50\%$$

b） M为合数时，这一问题为何更易出现？（提示：此时$\{ 0^2, 1^2, 2^2, \ldots \}$关于M的同余类更少）

【解答】

此时，对于秩$0 \le a < b < \lceil M/2 \rceil$，即便

$$a + b \equiv 0 \pmod{M}$$

$$a - b \equiv 0 \pmod{M}$$

均不成立，也依然可能有：

$$a^2 - b^2 = (a + b)\cdot(a - b) \equiv 0 \pmod{M}$$

作为实例，仍然考查以上长度M = 12的散列表，取a = 2和b = 4，则

$$2 + 4 = 6 \equiv 0 \pmod{12}$$

$$2 - 4 = -2 \equiv 0 \pmod{12}$$

均不成立，然而依然有：

$$2^2 - 4^2 = -12 \equiv 0 \pmod{12}$$

[9-16] 懒惰删除法尽管具有实现简明的优点，但随着装填因子的增大，查找操作的成本却将急剧上升。为克服这一缺陷，有人考虑在本章所给示例代码的基础上，做如下调整：

❶ 每次查找成功后，都随即将命中的词条前移至查找链中第一个带有懒惰删除标记的空桶（若的确存在且位于命中词条之前）

❷ 每次查找失败后，若查找链的某一后缀完全由带懒惰删除标记的空桶组成，则清除它们的标记

试问，这些方法是否可行？为什么？

【解答】

不可行。

这些方法均旨在及时地剔除带有懒惰删除标记的桶，实质上都等效于压缩查找链。设计者希望在花费一定时间做过这些处理之后，使得后续的查找得以加速，同时空间利用率也得以提高。

然而对于闭散列而言最大的难点在于：

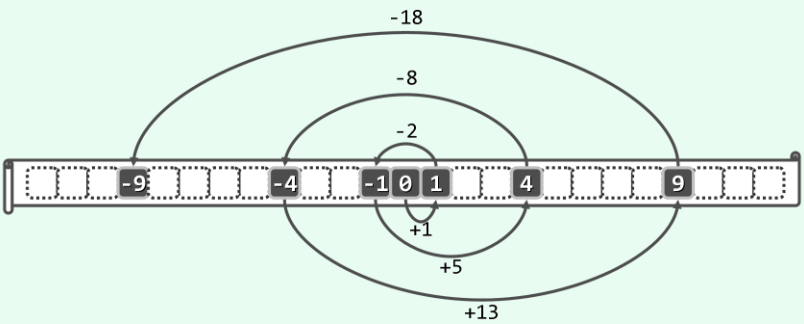
查找链可能彼此有所重叠，且
任何一个带有懒惰删除标记的桶，都可能同时属于多个查找链

因此其中某一条查找链的压缩，都将可能造成其它查找链的断裂。因此为使这些策略变得可行，还必须做更多的处理——但通常都未免弄巧成拙，得不偿失。

[9-17] 所谓双向平方试探法，是平方试探法的一种拓展变型。具体地如图 x9.2 所示，在出现冲突并需要排解时，将以

$$\{ +1^2, -1^2, +2^2, -2^2, +3^2, -3^2, +4^2, -4^2, \ldots \}$$

为间距依次试探。整个试探过程中，跳转的方向前、后交替，故此得名。



图x9.2 双向平方试探法

试证明，只要散列表长取作素数 M = 4k + 3（k 为非负整数），则：

a） 任一关键码所对应的查找链中，前 M 个桶必然互异（即取遍整个散列表）；

（提示：任一素数 M 可表示为一对整数的平方和，当且仅当 M ≡ 1 (mod 4)[①]）

【解答】

使用双向平方试探法，根据跳转的方向，查找链的前M项可以分为三类：

O：第1次试探，位于原地的起点
A：第2、4、6、...、M - 1次试探，相对于起点向前跳转
B：第3、5、7、...、M次试探，相对于起点向后跳转

根据习题[9-14]的结论，无论O∪A还是O∪B，其内部的试探均不致相互冲突。因此，只需考查A类与B类试探之间是否可能冲突。

假设第2a次（A类）试探，与第2b + 1次（B类）试探相互冲突。于是便有：

$$a^2 \equiv -b^2 \pmod{M}$$

亦即：

$$n = a^2 + b^2 \equiv 0 \pmod{M} \quad \cdots\cdots (*)$$

然而以下将证明，对于形如M = 4k + 3的素数表长，这是不可能的。

一个自然数n若可表示为一对整数的平方和，则称之为“可平方拆分的”。

不妨设n的素因子分解式为：

$$n = p_1^{\alpha 1} \cdot p_2^{\alpha 2} \cdot p_3^{\alpha 3} \cdot \ \cdots\ \cdot p_d^{\alpha d}$$

只要注意到以下恒等式：

$$(u^2 + v^2)(s^2 + t^2) = (us + vt)^2 + (ut - vs)^2$$

即不难理解，n是可平方拆分的，当且仅当对于每个$1 \le i \le d$，或者α_i是偶数，或者p_i是可平方拆分的。

除了$2 = 1^2 + 1^2$，其余的素数可以按照关于4的模余值，划分为两个同余类。而根据费马平方和定理，形如4k + 1的素因子必可平方拆分，而形如4k + 3的素因子则必不可平方拆分。因此n若可平方拆分，则对于其中每个形如$p_i = 4k + 3$的素因子，α_i必然是偶数。

现在，反观以上(*)式可知，M = 4k + 3应是n的一个素因子。而根据以上分析还可进一步推知，n必然能被M^2整除。于是便有：

$$n = a^2 + b^2 \ge M^2$$

然而，对于取值都在$[1, \lfloor M/2 \rfloor]$范围之内的a和b，这是不可能的。

b） 在装填因子尚未增至 100%之前，插入操作必然成功（而不致因无法抵达空桶而失败）。

【解答】

由以上分析结论，显然。

[①] 亦即，费马平方和定理（Two-Square Theorem of Fermat）

[9-18] 设散列表$\mathscr{H}$容量为 11 且初始为空，采用除余法确定散列地址，采用单向平方试探法排解冲突，采用懒惰策略实现删除操作。

a） 若通过 put()接口将关键码{ 2012, 10, 120, 175, 190, 230 }依次插入$\mathscr{H}$中，试给出此时各桶单元的内容（提示：仿照教材 274 页图 9.18）；

【解答】

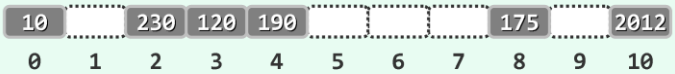
按照除余法，这一组关键码对应的初始试探位置依次为：

{ 10, 10, 10, 10, 3, 10 }

因此整个插入过程应该如下：

```
2012可直接存入10号桶
10经过2次试探，存入(10 + 1) % 11  =  0号桶
120经过3次试探，存入(10 + 1 + 3) % 11  =  3号桶
175经过4次试探，存入(10 + 1 + 3 + 5) % 11  =  8号桶
190经过2次试探，存入(3 + 1) % 11  =  4号桶
230经过6次试探，存入(10 + 1 + 3 + 5 + 7 + 9) % 11  =  2号桶
```

最终结果，如图x9.3所示。

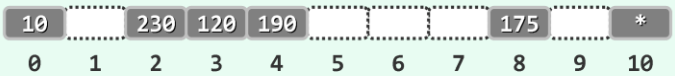


图x9.3 将关键码{ 2012, 10, 120, 175, 190, 230 }，依次插入长度为11的散列表

b） 若再执行 remove(2012)，试给出此时各桶单元的内容（提示：注意懒惰删除标记）；

【解答】

如图x9.4所示，10号桶被加注懒惰删除标记。



图x9.4 删除关键码2012，并做懒惰删除标记

c） 若继续执行 get(2012)，会出现什么问题？为什么？

（提示：此时虽只有 5 个关键码，但计入被懒惰删除的桶，等效的装填因子为(5 + 1)/11 > 50%）

【解答】

不难验证，形如：

$(10 + k^2) \% 11,\quad k = 0, 1, 2, 3, \ldots$

的整数只有6种选择：

{ 0, 2, 3, 4, 8, 10 }

它们构成了2012所对应的查找链。

然而反观如图x9.4所示的当前散列表可见，此时该查找链上所有的桶均非空余——其中5个存有关键码，1个带有懒惰删除标记。因此，对2012的查找必然陷入死循环。

d） 为避免此类问题的出现，可以采取什么措施？试给出至少两种方案。

【解答】

首先，在计算装填因子时，可以同时计入带有懒惰删除标记的桶。这样，一旦发现装填因子超过50%，则可通过重散列及时地扩容，令装填因子重新回落至50%以下——根据习题[9-14]的结论，如此即可保证，查找过程必然不致出现死循环。

另外，也可以改用习题[9-17]所建议的双向平方试探法排解冲突。当然，此后如需通过重散列扩容，则散列表的容量M必须与初始的11一样，依然是形如4k + 3的素数。

[9-19] a） 试在图结构的邻接表实现方式中，将每一列表替换为散列表；

【解答】

请读者独立完成编码和调试任务。

b） 如此，图 ADT 各操作接口的时间复杂度有何变化？

【解答】

请读者结合自己的具体实现方法，给出分析结论。

c） 总体空间复杂度有何变化？

【解答】

请读者针对自己的具体实现方法，给出分析结论。

[9-20] a） 了解 C#所提供 GetHashCode()方法的原理，并尝试利用该方法转换散列码；

【解答】

请读者独立完成代码阅读及相关实验任务。

b） 了解 Java 所提供 hashCode()方法的原理，并尝试利用该方法转换散列码；

【解答】

对于任意对象x，x.hashCode()返回的散列码实际上就是x在内存中的地址。因此，可能出现如下奇特的现象：（包括关键码在内）数值完全相等的两个对象，散列码居然不同。

请读者独立完成代码阅读及相关实验任务。

c） 这两个接口存在什么潜在的问题？为此在实际应用中，还需对它们做何调整？

【解答】

请读者独立完成代码阅读及分析任务，并根据自己的理解给出解答。

[9-21] 考查教材 9.4.1 节介绍的基本桶排序算法。

若采用习题[9-11]中的技巧，可将其中散列表初始化所需的时间从 $O(M)$优化至常数。

a） 算法的整体时间复杂度，是否因此亦有所改进？

【解答】

因为最后一步仍然需要花费$O(M)$时间遍历整个散列表，故总体的渐进时间复杂度并无改变。

b） 空间方面，需要付出多大的代价？是否会影响到渐进的空间复杂度？

【解答】

按照习题[2-34]中Bitmap类的实现方式，新增的空间与原先所需的空间渐进等量，故亦总体的渐进空间复杂度亦保持不变。

[9-22] 任给来自于$[0, n^d)$范围内的 n 个整数，其中常数 d > 1。

试设计并实现一个算法，在$O(n)$时间内完成对它们的排序。（提示：基数排序）

【解答】

首先，在$O(dn) = O(n)$时间内，将这些整数统一转换为n进制的表示。如此，每个整数均不超过d位。若将每一位视作一个域（字段），则这些整数的排序依据，即等效于（由高位至低位）按照这些域的字典序。因此接下来，只需直接套用基数排序算法，即可实现整体排序。

以上基数排序过程包含d趟桶排序，累计耗时：

$$d \cdot O(n) = O(dn) = O(n)$$

[9-23] 若将任一有序序列等效地视作有序向量，则其中每个元素的秩，应恰好就等于序列中不大于该元素的元素总数。例如，其中最小、最大元素的秩分别为 0、n - 1，可以解释为：分别有 0 和 n - 1 个元素不大于它们。根据这一原理，只需统计出各元素所对应的这一指标，也就确定了它们在有序向量中各自所对应的秩。

a） 试按照以上思路，实现一个排序算法[②]；

【解答】

表x9.2 对序列{ 5a, 2a, 3, 2b, 9a, 5b, 9b, 8, 2c }的直接计数排序

输入序列	5a	2a	3	2b	9a	5b	9b	8	2c
更小的元素总数	4	0	3	0	7	4	7	6	0
相等的前驱总数	0	0	0	1	0	1	1	0	2
在排序序列中的秩	4	0	3	1	7	5	8	6	2

为每个元素设置一个计数器，初始值均取作0。以下对于每个元素，都遍历一趟整个序列，并统计出小于该元素的元素总数，以及在位于该元素之前、与之相等的元素总数。最后，根据以上两项之和，即可确定各元素在排序序列中对应的秩。

[②] 亦即，所谓的计数排序（counting sort）算法

仍如教材277页图9.21所示，考查待排序序列{ 5a, 2a, 3, 2b, 9a, 5b, 9b, 8, 2c } 按照以上算法，每个元素的各项统计数值如表x9.2所示。

对照教材的图9.21可见，排序结果完全一致。

请特别留意这里选择的扫描方向，并体会为何如此可以保证该算法的稳定性。

b） 你的这一算法，时间和空间复杂度各是多少？

【解答】

该算法供需$O(n)$趟遍历，每趟遍历均需$O(n)$时间，故累计耗时为$O(n^2)$。

除了原输入序列，这里引入的计数器还共需$O(n)$辅助空间。

c） 改进你的算法，使之能够在 $O(n + M)$时间内对来自[0, M)范围内的 n 个整数进行排序，且使用的辅助空间不超过 $O(M)$。

【解答】

计算过程，大致可以描述如算法x9.1所示：

```
int* countingSort(int A[0, n))
   引入一个可计数的散列表H[0, M)，其长度等于输入元素取值范围的宽度M
   将H[]中所有桶的数值，初始化为0
   遍历输入序列A[0, n)  //遍历计数，O(n)
      对于每一项A[k]，令H[ A[k] ] ++
   遍历散列表H[0, M)  //逐项累加，O(M)
      对于每一项H[i]，令H[ i + 1 ] += H[i]
   创建序列S[0, n)，用以记录排序结果
   逆向遍历输入序列A[0, n)  //逐项输出，O(n)
      对于每一项A[k]
         令S[ -- H[ A[k] ] ] = A[k]
   返回S[0, n)
```

算法x9.1 整数向量的计数排序算法

同样地，也请读者特别留意这里对输入序列和散列表的扫描方向，并体会为何如此可以保证该算法的稳定性。

其中每个步骤各自所需的时间，如注释所示。总体而言，执行时间不超过$O(n + M)$。需要特别说明的是，若n >> M，则排序时间为$O(n)$，优于面向一般情况最优的$O(n\log n)$。另外从算法流程这也就是所谓的“小集合、大数据”情况，在当下实际应用中，这已成为数据和信息处理的主流需求类型。

空间方面，除了输出序列S[]，这里只引入了一个规模为$O(M)$的散列表。

仍以教材277页图9.21中序列为例。按照以上算法，所有元素各项统计数值如表x9.3所示。

表x9.3 借助散列表对{ 5a, 2a, 3, 2b, 9a, 5b, 9b, 8, 2c }的计数排序（凡“-”项均与其上方项相等）

k		0	1	2	3	4	5	6	7	8	9	输出
$\mathcal{H}$[k]	$\mathcal{H}$[k]初始值	0	0	0	0	0	0	0	0	0	0	
	遍历计数之后	0	0	3	1	0	2	0	0	1	2	
	逐项累加之后	0	0	3	4	4	6	6	6	7	9	
	逆序逐项输出	-	-	2	-	-	-	-	-	-	-	S[2] = 2c
		-	-	-	-	-	-	-	-	6	-	S[6] = 8
		-	-	-	-	-	-	-	-	-	8	S[8] = 9b
		-	-	-	-	-	5	-	-	-	-	S[5] = 5b
		-	-	-	-	-	-	-	-	-	7	S[7] = 9a
		-	-	1	-	-	-	-	-	-	-	S[1] = 2b
		-	-	-	3	-	-	-	-	-	-	S[3] = 3
		-	-	0	-	-	-	-	-	-	-	S[0] = 2a
		-	-	-	-	-	4	-	-	-	-	S[4] = 5a

[9-24] 习题[4-18]（100 页）曾指出，同一整数可能同时存在多个费马-拉格朗日（Fermat-Lagrange）分解，其中，四个整数之和最小者称作最小分解。比如：

$$101 = 0^2 + 0^2 + 1^2 + 10^2 = (0, 0, 1, 10)$$
$$= 0^2 + 1^2 + 6^2 + 8^2 = (0, 1, 6, 8)$$
$$= 0^2 + 2^2 + 4^2 + 9^2 = (0, 2, 4, 9)$$
$$= 0^2 + 4^2 + 6^2 + 7^2 = (0, 4, 6, 7)$$
$$= 2^2 + 5^2 + 6^2 + 6^2 = (2, 5, 6, 6)$$

其中(0, 0, 1, 10)即为 101 的最小费马-拉格朗日分解，因为组成它的四个整数之和 11 为最小。

a） 试设计并实现一个算法，对任何整数 n > 0，输出[1, n]内所有整数的最小费马-拉格朗日分解；

【解答】

引入散列表$\mathcal{H}$[0, n)，记录此区间内各整数当前的最小分解方案。枚举所有可能的分解方案，并不断刷新各散列表项。当然，在枚举的过程中，需充分利用该问题的特点，做有效的剪枝。

请读者根据以上提示，独立完成编码和调试任务。

b） 你的算法需要运行多少时间？空间呢？

【解答】

蛮力算法大致需要运行$\mathcal{O}((\sqrt{n})^4) = \mathcal{O}(n^2)$时间；空间主要消耗于散列表，占用$\mathcal{O}(n)$的辅助空间。请读者根据各自所设计并采用的优化策略，给出更加具体和准确的估计。

[9-25] 散列技术在信息加密领域有着广泛应用，比如数字指纹的提取与验证。试通过查阅资料和编程实践：

a） 了解 MD5、SHA 等主流数字指纹的定义、功能、原理及算法流程；

【解答】

请读者独立完成相关资料的阅读，以及算法的编码和调试任务。

b） 以 Python 语言提供的 hashlib 模块库为例，学习 md5()、sha1()、sha224()、sha256()、sha384()、sha512()等接口的使用方法。

【解答】

请读者独立完成相关资料的阅读，并学习相关接口的使用方法。

[9-26] 当元素类型为字符串时，为避免复杂的散列码转换，可以改用键树（trie）结构来实现词典 ADT。

a） 试在 118 页习题[5-30]的基础上，基于键树结构实现词典的 get()、put()和 remove()接口，要求其时间复杂度分别为 $O(h)$、$O(hr)$和 $O(hr)$，其中 h 为树高，$r = |\Sigma|$为字符表规模。

【解答】

请读者根据有关介绍及提示，独立完成编码和调试任务。

b） remove()接口复杂度中的因子 r 可否消除？（提示：之所以会有因子 r，是因为在最坏情况下，在删除每个节点之前，都需要花费 $O(r)$的时间，确认对应向量中的每个指针是否都是 NULL）

【解答】

若沿用习题[5-30]的方式，用向量实现每个节点，则正如以上提示所指出的原因，无法消除remove()接口复杂度中的因子r。

c） put()接口复杂度中的因子 r 可否消除？（提示：之所以会有因子 r，是因为在最坏情况下，在创建每个节点之后，都需要花费 $O(r)$的时间，将对应向量中的每个指针都初始化为 NULL）

【解答】

与b）同理，若用向量实现键树节点，则put()接口复杂度中的因子r亦难以消除。

d） 试举例说明，以上实现方式在最坏情况下可能需要多达$\Omega(nr)$的空间，其中 $n = |S|$为字符串集的规模。

【解答】

比如，若S中的字符串均互不为前缀，则每个字符串都唯一对应于一个叶节点。于是，即便只计入这n个叶节点，累计空间总量也至少有$n \cdot \Omega(r) = \Omega(nr)$。

e） 试改用列表来实现各节点，使所需空间的总量线性正比于 S 中所有字符串的长度总和——当然，get()接口的效率因此会降至 $O(hr)$，其中 h 为树高，同时也是 S 中字符串的最大长度。（提示：参考文献[54]）

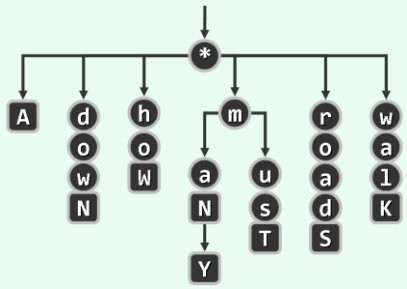
【解答】

改用列表实现各节点后，每个节点的规模与实际的分支数成正比，每个字符串的每个字符至多占用$O(1)$的空间，总体空间消耗量不超过所有字符串的总长。

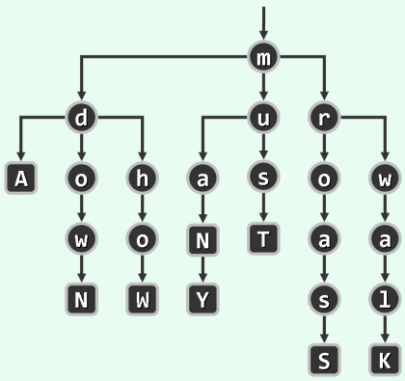
为此，在每个节点需要$O(r)$时间做顺序查找，以确定深入的分支方向，总体时间增至$O(hr)$。

请读者根据以上介绍和提示，并参考建议的文献，独立完成编码和调试任务。

f） 键树中往往包含大量的单分支节点。试如图 x9.5 所示，通过折叠合并相邻的单分支节点，进一步提高键树的时、空效率。改进之后，键树的时、空复杂度各是多少？（提示：参考文献[55]）



图x9.5 PATRICIA树（PATRICIA tree）[③]



图x9.6 三叉键树（ternary trie）[④]

【解答】

具体地，也就是将向量的单分支节点合成一个大节点。尽管可以在一定程度上提高时、空效率，但从渐进角度看并无实质改进。

请读者根据以上介绍和提示，并参考建议的文献，独立完成编码和调试任务。

g） 习题[8-19]（173 页）曾介绍过四叉树（quadtree）结构，并指出其深度不受限制的缺陷。若将四个象限的二进制编码视作字符，即将字符表取作 Σ = { 00, 01, 10, 11 }，则四叉树可以看作键树的特例。试基于这一理解，仿照以上技巧对四叉树进行压缩，使其深度不致超过 $O(n)$。

【解答】

同样地，对于如此表示的四叉树，可以将其中相邻的单分支节点合并为大节点。

如此压缩之后，叶节点的总数固然不超过输入点集的规模n，同时内部节点也不会超过n，故总体的深度可以控制在$O(n)$范围以内。

h） 仿照教材 5.1.3 节将有根有序多叉树等价变换为二叉树的技巧，试如图 x9.6 所示，以三叉树的形式进一步改进键树。其中，任一节点 x 的左、中、右分支非空，当且仅当 S 中存在下一字符小于、等于、大于 x 的字符串。以图中深度为 1 的节点 u 为例：其左分支非空，是因为 S 中存在首字符为 m、次字符小于 u 字符串（"man"和"many"）；反之，其右分支为空，是因为 S 中不存在首字符为 m、次字符大于 u 的字符串。改进之后，键树的时、空复杂度各是多少？
（提示：参考文献[56]）

【解答】

请读者根据以上介绍和提示，独立完成编码和调试任务，并针对具体的实现方式分析复杂度。

③ 由D. Morrison于1968年发明[55]
名字源自 “Practical Algorithm To Retrieve Information Coded In Alphanumeric” 的缩写

④ 由J. Bentley和R. Sedgewick于1997年发明[56]