# 第10章 索引与散列

**一、复习要点**

索引结构和散列结构是用于外部搜索的搜索结构。数据在外存的组织即文件结构，主要分顺序、直接存取（散列）和索引文件。在这些文件组织中使用的主要是索引和散列方法。

**1、**基本知识点

要求掌握静态索引结构，包括线性索引、倒排索引、静态索引树的搜索和构造方法。掌握动态索引结构，包括**B**树的搜索、插入、删除，通过关键码个数估算**B**树的高度的方法；**B+**树的搜索、插入与删除。掌握散列法，包括散列函数的构造、处理溢出的闭散列方法；处理溢出的开散列方法；散列表分析。

**二、难点与重点**

1、线性索引

⮚ 密集索引、稀疏索引、索引表计算

⮚ 基于属性查找建立倒排索引、单元式倒排表

2、动态搜索树

⮚ 平衡的m路搜索树的定义、搜索算法

⮚ B树的定义、B树与平衡的m路搜索树的关系

* B树的插入(包括结点分裂)、删除(包括结点调整与合并)方法

⮚ B树中结点个数与高度的关系

⮚ B+树的定义、搜索、插入与删除的方法

3、散列表

* 散列函数的比较
* 装载因子 α 与平均搜索长度的关系，平均搜索长度的关系
* 表长m、表中已有数据对象个数n和装载因子的关系
* 解决冲突的(闭散列)线性探查法的运用，平均探查次数的计算
* 线性探查法的删除问题、散列表类的设计中必须为各地址设置三个状态
* 线性探查法中的堆积聚集问题
* 解决冲突的(闭散列)双散列法的运用，平均探查次数计算
* 双散列法中再散列函数的设计要求与表长m互质，为此m设计为质数较宜
* 解决冲突的(闭散列)二次散列法的运用，平均探查次数计算
* 注意：二次散列法中装载因子α与表长m的设置
* 解决冲突的(开散列)开散列法的运用，平均探查次数计算
* 由平均探查次数计算装载因子α，再计算表大小的方法

**三、教材中习题的解析**

10-1 什么是静态索引结构？什么是动态索引结构？它们各有哪些优缺点？

【解答】

静态索引结构指这种索引结构在初始创建，数据装入时就已经定型，而且在整个系统运行期间，树的结构不发生变化，只是数据在更新。动态索引结构是指在整个系统运行期间，树的结构随数据的增删及时调整，以保持最佳的搜索效率。静态索引结构的优点是结构定型，建立方法简单，存取方便；缺点是不利于更新，插入或删除时效率低。动态索引结构的优点是在插入或删除时能够自动调整索引树结构，以保持最佳的搜索效率；缺点是实现算法复杂。

10-2 设有10000个记录对象, 通过分块划分为若干子表并建立索引, 那么为了提高搜索效率, 每一个子表的大小应设计为多大?

【解答】

每个子表的大小 s = ⎡n⎤ = ⎡10000⎤ = 100 个记录对象。

10-3如果一个磁盘页块大小为1024 (=1K) 字节，存储的每个记录对象需要占用16字节，其中关键码占4字节，其它数据占12字节。所有记录均已按关键码有序地存储在磁盘文件中，每个页块的第1个记录用于存放线性索引。另外在内存中开辟了256K字节的空间可用于存放线性索引。试问：

(1) 若将线性索引常驻内存，文件中最多可以存放多少个记录？(每个索引项8字节，其中关键码4字节，地址4字节)

(2) 如果使用二级索引，第二级索引占用1024字节（有128个索引项），这时文件中最多可以存放多少个记录？

【解答】

(1) 因为一个磁盘页块大小为1024字节，每个记录对象需要占用16字节，则每个页块可存放1024 / 16 = 64个记录，除第一个记录存储线性索引外，每个页块可存储63个记录对象。又因为在磁盘文件中所有记录对象按关键码有序存储，所以线性索引可以是稀疏索引，每一个索引项存放一个页块的最大关键码及该页块的地址。若线性索引常驻内存，那么它最多可存放256 \* (1024 / 8 ) = 256 \* 128 = 32768个索引项，文件中可存放 32768 \* 63 = 2064384个记录对象。

(2) 由于第二级索引占用1024个字节，内存中还剩255K 字节用于第一级索引。第一级索引有255 \* 128 = 32640个索引项，作为稀疏索引，每个索引项索引一个页块，则索引文件中可存放32640 \* 63 = 2056320。

|  |  |
| --- | --- |
| 397 | Hello World! |
| 82 | XYZ |
| 1038 | This string is rather long |
| 1037 | This is Shorter |
| 42 | ABC |
| 2222 | Hello new World! |

10-4 假设在数据库文件中的每一个记录是由占2个字节的整型数关键码和一个变长的数据字段组成。数据字段都是字符串。为了存放右面的那些记录，应如何组织线性索引？

【解答】

将所有字符串依加入的先后次序存放于一个连续的存储空间store中，这个空间也叫做“堆”，它是存放所有字符串的顺序文件。它有一个指针free，指示在堆store中当前可存放数据的开始地址。初始时free置为0，表示可从文件的0号位置开始存放。线性索引中每个索引项给出记录关键码，字符串在store中的起始地址和字符串的长度：

索引表ID 堆store

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 关键码 | 串长度 | 串起始地址 | 0 | Hello World! XYZ This string is rather long This |
| 42 | 3 | 56 |  |  |
| 82 | 3 | 12 |  | is Shorter ABC Hello new World! |
| 397  free | 12 | 0 |  |  |
| 1037 | 15 | 41 |  |  |
| 1038 | 26 | 15 |  |  |
| 2222 | 16 | 59 |  |  |

10-5 设有一个职工文件：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 记录地址 | 职工号 | 姓 名 | 性别 | 职 业 | 年龄 | 籍贯 | 月工资(元) |
| 10032 | 034 | 刘激扬 | 男 | 教 师 | 29 | 山东 | 720.00 |
| 10068 | 064 | 蔡晓莉 | 女 | 教 师 | 32 | 辽宁 | 1200.00 |
| 10104 | 073 | 朱 力 | 男 | 实验员 | 26 | 广东 | 480.00 |
| 10140 | 081 | 洪 伟 | 男 | 教 师 | 36 | 北京 | 1400.00 |
| 10176 | 092 | 卢声凯 | 男 | 教 师 | 28 | 湖北 | 720.00 |
| 10212 | 123 | 林德康 | 男 | 行政秘书 | 33 | 江西 | 480.00 |
| 10248 | 140 | 熊南燕 | 女 | 教 师 | 27 | 上海 | 780.00 |
| 10284 | 175 | 吕 颖 | 女 | 实验员 | 28 | 江苏 | 480.00 |
| 10320 | 209 | 袁秋慧 | 女 | 教 师 | 24 | 广东 | 720.00 |

其中，关键码为职工号。试根据此文件，对下列查询组织主索引和倒排索引，再写出搜索结果关键码。(1) 男性职工；(2) 月工资超过800元的职工；(3) 月工资超过平均工资的职工；(4) 职业为实验员和行政秘书的男性职工；(5) 男性教师或者年龄超过25岁且职业为实验员和教师的女性职工。

【解答】

主索引 月工资 倒排索引 职务 倒排索引

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 职工号 | 记录地址 |  | 月工资 | 长度 | 指针 |  | 职务 | 长度 | 指针 |  |
| 0 | 034 | 10032 |  | 480. | 3 | 073 |  | 教师 | 6 | 034 |  |
| 1 | 064 | 10068 |  |  |  | 123 |  |  |  | 064 |  |
| 2 | 073 | 10104 |  |  |  | 175 |  |  |  | 081 |  |
| 3 | 081 | 10140 |  | 720. | 3 | 034 |  |  |  | 092 |  |
| 4 | 092 | 10176 |  |  |  | 092 |  |  |  | 140 |  |
| 5 | 123 | 10212 |  |  |  | 209 |  |  |  | 209 |  |
| 6 | 140 | 10248 |  | 780. | 1 | 140 |  | 实验员 | 2 | 073 |  |
| 7 | 175 | 10284 |  | 1200. | 1 | 064 |  |  |  | 175 |  |
| 8 | 209 | 10320 |  | 1400. | 1 | 081 |  | 行政秘书 | 1 | 123 |  |

性别 倒排索引 年龄 倒排索引

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 性别 | 长度 | 指针 |  | 年龄 | 长度 | 指针 |
| 男 | 5 | 034 |  | 24 | 1 | 209 |
|  |  | 073 |  | 26 | 1 | 073 |
|  |  | 081 |  | 27 | 1 | 140 |
|  |  | 092 |  | 28 | 2 | 092 |
|  |  | 123 |  |  |  | 175 |
| 女 | 4 | 064 |  | 29 | 1 | 034 |
|  |  | 140 |  | 32 | 1 | 064 |
|  |  | 175 |  | 33 | 1 | 123 |
|  |  | 209 |  | 36 | 1 | 081 |

搜索结果：

(1) 男性职工 (搜索性别倒排索引)：**{**034, 073, 081, 092, 123**}**

(2) 月工资超过800元的职工 (搜索月工资倒排索引)：**{**064, 081**}**

(3) 月工资超过平均工资的职工(搜索月工资倒排索引) **{**月平均工资776元**}**：

**{**064, 081, 140**}**

(4) 职业为实验员和行政秘书的男性职工(搜索职务和性别倒排索引)：

**{**073, 123, 175**}** **&&** **{**034, 073, 081, 092, 123**}** = **{**073, 123**}**

(5) 男性教师 (搜索性别与职务倒排索引)：

**{**034, 073, 081, 092, 123**} && {** 034, 064, 081, 092, 140, 209**} = {**034, 081, 092**}**

年龄超过25岁且职业为实验员和教师的女性职工 (搜索性别、职务和年龄倒排索引)：

**{**064, 140, 175, 209**} && {**034, 064, 073, 081, 092, 140, 175, 209**} && {**034, 064, 073, 081,092, 123, 140, 175**}** = **{**064, 140, 175**}**

10-6 倒排索引中的记录地址可以是记录的实际存放地址，也可以是记录的关键码。试比较这两种方式的优缺点。

【解答】

在倒排索引中的记录地址用记录的实际存放地址，搜索的速度快；但以后在文件中插入或删除记录对象时需要移动文件中的记录对象，从而改变记录的实际存放地址，这将对所有的索引产生影响：修改所有倒排索引的指针，不但工作量大而且容易引入新的错误或遗漏，使得系统不易维护。

记录地址采用记录的关键码，缺点是寻找实际记录对象需要再经过主索引，降低了搜索速度；但以后在文件中插入或删除记录对象时，如果移动文件中的记录对象，导致许多记录对象的实际存放地址发生变化，只需改变主索引中的相应记录地址，其他倒排索引中的指针一律不变，使得系统容易维护，且不易产生新的错误和遗漏。

10-7 m = 2的平衡m路搜索树是AVL树，m = 3的平衡m路搜索树是2-3树。它们的叶结点必须在同一层吗？m阶B树是平衡m路搜索树，反过来，平衡m路搜索树一定是B树吗？为什么？

【解答】

m = 3的平衡m路搜索树的叶结点不一定在同一层，而m阶B\_树的叶结点必须在同一层，所以m阶B\_树是平衡m路搜索树，反过来，平衡m路搜索树不一定是B\_树。

10-8 下图是一个3阶B树。试分别画出在插入65、15、40、30之后B树的变化。

55

80 90

45

95

85

60 70

50

25 35

【解答】

插入65后：

55 80

90

65

45

25 35

50

60

70

85

95

插入15后：

55 80

90

65

25 45

15

70

60

95

85

50

35

插入40后：

55 80

90

65

25 45

50

35 40

15

70

60

95

85

插入30后：

55

80

35

45

25

90

65

30

15

40

70

60

95

85

50

10-9 下图是一个3阶B树。试分别画出在删除50、40之后B树的变化。

50

30

60 80

55

20

95

70

40

【解答】

删除50后：

55

80

30

95

60 70

40

20

删除40后：

55 80

20 30

60 70

95

10-10 对于一棵有1999999个关键码的199阶B树，试估计其最大层数(不包括失败结点)及最小层数(不包括失败结点)。

【解答】

设B树的阶数m = 199，则⎡m/2⎤ = 100。若不包括失败结点层，则其最大层数为

⎣log⎡m/2⎤ ((N+1)/2)⎦ = ⎣log100 1000000⎦ = 3

若使得每一层关键码数达到最大，可使其层数达到最小。第0层最多有 (m-1) 个关键码，第1层最多有m(m-1) 个关键码，第2层最多有 m2 (m-1) 个关键码，…，第h-1层最多有mh-1 (m-1) 个关键码。层数为h的B树最多有 (m-1) + m(m-1) + m2 (m-1) + … + mh-1 (m-1) = (m-1) (mh–1) / (m-1) = mh–1个关键码。反之，若有n个关键码，n≤mh–1，则 h ≥ log m (n+1)，所以，有1999999个关键码的199阶B树的最小层数为

⎡log m (n+1)⎤ = ⎡log199 (1999999 +1)⎤ = ⎡log 199 2000000⎤ = 3

10-11 给定一组记录，其关键码为字符。记录的插入顺序为 **{** C, S, D, T, A, M, P, I, B, W, N, G, U, R, K, E, H, O, L, J **}**，给出插入这些记录后的4阶B+树。假定叶结点最多可存放3个记录。

【解答】

插入C, S, D 插入T 插入A 插入M

D S

S

S

C D S

D M

S T

A C

A C D

S T

S T

C D

插入P 插入I

D M S

D S

M P

D I

S T

A C

D M P

A C

S T

插入B, W, N, G 插入U

D M S U

D M S

U W

S T

M N P

D G I

A B C

D G I

M N P

S T W

A B C

插入R

P

S U

D M

M N

D G I

A B C

P R

S T

U W

插入K

P

S U

D I M

I K

D G

M N

P R

S T

U W

A B C

插入E

P

S U

D I M

D E G

I K

S T

P R

U W

M N

A B C

插入H

P

D G I M

S U

G H

I K

M N

S T

P R

U W

D E

A B C

插入O, L

P

D G I M

S U

M N O

A B C

I K L

S T

P R

G H

D E

U W

插入J

I P

D G

K M

S U

I J

K L

P R

S T

M N O

U W

G H

D E

A B C

10-12 设有一棵B+树，其内部结点最多可存放100个子女，叶结点最多可存储15个记录。对于1, 2, 3, 4, 5层的B+树，最多能存储多少记录，最少能存储多少记录。

【解答】

一层B+树：根据B+树定义，一层B+树的结点只有一个，它既是根结点又是叶结点，最多可存储m1 = 15个记录，最少可存储 ⎡m1/2⎤ = 8个记录。

二层B+树：第0层是根结点，它最多有m = 100棵子树，最少有2个结点；第1层是叶结点，它最多有m个结点，最多可存储m\*m1 = 100\*15 = 1500个记录，最少有2个结点，最少可存储2\* ⎡m1/2⎤ = 16个记录。

三层B+树：第2层是叶结点。它最多有m2个结点，最多可存储m2 \* m1 = 150000个记录。最少有2\* ⎡m/2⎤ = 100个结点，最少可存储2\* ⎡m/2⎤ \* ⎡m1/2⎤ = 800个记录。

四层B+树：第3层是叶结点。它最多有m3个结点，可存储m3 \* m1 = 15000000个记录。最少有2\* ⎡m/2⎤ 2 = 2 \* 502 = 5000个结点，存储2\* ⎡m/2⎤ 2 \* ⎡m1/2⎤ = 40000个记录。

五层B+树：第4层是叶结点。它最多有m4个结点，可存储m4 \* m1 = 1500000000个记录。最少有2\* ⎡m/2⎤ 3 = 2 \* 503 = 250000个结点，存储2\* ⎡m/2⎤ 3 \* ⎡m1/2⎤ = 2000000个记录。

10-13设散列表为HT[13], 散列函数为 H (*key*) = *key* %13。用闭散列法解决冲突, 对下列关键码序列 12, 23, 45, 57, 20, 03, 78, 31, 15, 36 造表。

(1) 采用线性探查法寻找下一个空位, 画出相应的散列表, 并计算等概率下搜索成功的平均搜索长度和搜索不成功的平均搜索长度。

(2) 采用双散列法寻找下一个空位, 再散列函数为 RH (*key*) = (7\**key*) % 10 + 1, 寻找下一个空位的公式为 Hi = (Hi-1 + RH (*key*)) % 13, H1 = H (*key*)。画出相应的散列表, 并计算等概率下搜索成功的平均搜索长度。

【解答】

使用散列函数 H(*key*) = *key* **mod** 13，有

H(12) = 12, H(23) = 10, H(45) = 6, H(57) = 5,

H(20) = 7, H(03) = 3, H(78) = 0, H(31) = 5,

H(15) = 2, H(36) = 10.

(1) 利用线性探查法造表：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 78 |  | 15 | 03 |  | 57 | 45 | 20 | 31 |  | 23 | 36 | 12 |
| (1) |  | (1) | (1) |  | (1) | (1) | (1) | (4) |  | (1) | (2) | (1) |

搜索成功的平均搜索长度为

ASLsucc = (1 + 1 + 1 + 1 + 1 + 1 + 4 + 1 + 2 + 1) = 

搜索不成功的平均搜索长度为

ASLunsuc*c*= (2 + 1 + 3 + 2 + 1 + 5 + 4 + 3 + 2 + 1 + 5 + 4 + 3) = 

(2) 利用双散列法造表：

Hi = (Hi-1 + RH (key)) %13, H1 = H (key)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 78 |  | 15 | 03 |  | 57 | 45 | 20 | 31 | 36 | 23 |  | 12 |
| (1) |  | (1) | (1) |  | (1) | (1) | (1) | (3) | (5) | (1) |  | (1) |

搜索成功的平均搜索长度为

ASLsucc = (1 + 1 + 1 + 1 + 1 + 1 + 3 + 5 + 1 + 1) = 

10-14 设有150个记录要存储到散列表中, 要求利用线性探查法解决冲突, 同时要求找到所需记录的平均比较次数不超过2次。试问散列表需要设计多大？ 设α是散列表的装载因子，则有



【解答】

已知要存储的记录数为n = 150，查找成功的平均查找长度为ASLsucc ≤ 2，则有ASLsucc =≤ 2，解得 α ≤。又有α = ≤ ，则 m ≥ 225。

10-15 若设散列表的大小为m，利用散列函数计算出的散列地址为h = hash(x)。

(1) 试证明：如果二次探查的顺序为(h + q2), (h + (q-1)2), …, (h+1), h, (h-1), …, (h-q2)，其中， q = (m-1)/2。因此在相继被探查的两个桶之间地址相减所得的差取模(%m)的结果为

m-2, m-4, m-6, …, 5, 3, 1, 1, 3, 5, …, m-6, m-4, m-2

(2) 编写一个算法，使用课文中讨论的散列函数h(x)和二次探查解决冲突的方法，按给定值x来搜索一个大小为m的散列表。如果x不在表中，则将它插入到表中。

【解答】

1. 将探查序列分两部分讨论：

(h + q2), (h + (q-1)2), …, (h+1), h 和 (h-1), (h-22), …, (h-q2)。

对于前一部分，设其通项为h + ( q – d )2, d = 0, 1, …, q，则相邻两个桶之间地址相减所得的差取模：

( h + (q – (d -1) )2 – ( h + (q – d )2 ) ) % m = ( (q – (d -1 ) )2 – (q – d )2 ) % m

= (2\*q -2\*d +1) % m = ( m – 2\*d ) % m. ( 代换 q = (m-1)/2 )

代入 d = 1, 2, …, q，则可得到探查序列如下：

m-2, m-4, m-6, …, 5, 3, 1。 ( m – 2\*q = m – 2\* (m-1)/2 = 1 )

对于后一部分，其通项为h – ( q – d )2, d = q, q+1, …, 2q，则相邻两个桶之间地址相减所得的差取模：

( h – ( q – d )2 – ( h – ( q – (d+1) )2 ) ) % m = ( ( q – (d+1)2 – (q – d )2 ) % m

= ( 2\*d – 2\*q +1) % m = ( 2\*d – m + 2) % m ( 代换 q = (m-1)/2 )

代入 d = q, q+1, …, 2q-1，则可得到

2\*d–m+2 = 2\*q – m +2 = m – 1 – m +2 = 1,

2\*d–m+2 = 2\*q + 2 – m +2 = m – 1 + 2 – m +2 = 3, ……,

2\*d–m+2 = 2\*(2\*q-1) – m +2 = 2\*(m–1–1) – m + 2 = 2\*m – 4 – m +2 = m – 2。〖证毕〗

(2) 编写算法

下面是使用二次探查法处理溢出时的散列表类的声明。

**template <class Type> class** HashTable **{** //散列表类的定义

**public:**

**enum** KindOfEntry **{** Active, Empty, Deleted **};** //表项分类 (活动 / 空 / 删)

HashTable ( ) **:** TableSize ( DefaultSize ) **{** AllocateHt ( )**;** CurrentSize = 0**; }** //构造函数

~HashTable ( ) **{ delete** [ ] ht**;** **}** //析构函数

**const** HashTable **& operator** = ( **const** HashTable **&** ht2 )**;** //重载函数：表赋值

7 **int** Find ( **const Type &** x )**;** //在散列表中搜索*x*

**int** IsEmpty ( ) **{ return** !CurrentSize **?** 1 **:** 0**;** **}** //判散列表空否，空则返回1

**private:**

**struct** HashEntry **{**  //散列表的表项定义

**Type** Element**;** //表项的数据, 即表项的关键码

KindOfEntry info**;** //三种状态: Active, Empty, Deleted

HashEntry ( ) **:** info (Empty ) **{ }** //表项构造函数

HashEntry ( **const Type &**E, KindOfEntry i = Empty ) **:** Element (E), info (i) **{ }**

**};**

**enum {** DefualtSize = 31**;** **}**

HashEntry \*ht**;** //散列表存储数组

**int** TableSize**;** //数组长度，是满足4k+3的质数，k是整数

**int** CurrentSize**;** //已占据散列地址数目

**void** AllocateHt ( ) **{** ht = **new** HashEntry[TableSize ]**;** **}** //为散列表分配存储空间**;**

**int** FindPos ( **const Type &** x )**;** //散列函数

**};**

**template <class Type> const** HashTable**<Type> &** HashTable**<Type> ::**

**operator** = ( **const** HashTable**<Type>** **&**ht2 ) **{**

//重载函数：复制一个散列表ht2

**if** ( **this** != **&**ht2 ) **{**

**delete** [ ] ht**;** TableSize = ht2.TableSize**;** AllocateHt ( )**;** //重新分配目标散列表存储空间

**for** ( **int** i = 0**;** i < TableSize**;** i++ ) ht[i] = ht2.ht[i]**;** //从源散列表向目标散列表传送

CurrentSize = ht2.CurrentSize**;** //传送当前表项个数

**}**

**return** \***this;** //返回目标散列表结构指针

**}**

**template <class Type> int** HashTable**<Type> ::** Find ( **const Type&** x ) **{**

//共有函数： 找下一散列位置的函数

**int** i = 0**,** q = ( TableSize -1 ) / 2, h0**;** // i为探查次数

**int** CurrentPos = h0 = HashPos ( x )**;** //利用散列函数计算x的散列地址

**while** ( ht[CurrentPos].info != Empty **&&** ht[CurrentPos].Element != x ) **{**

/搜索是否要求表项

**if** ( i <= q ) **{** //求“下一个”桶

CurrentPos = h0 + (q - i ) \* ( q - i )**;**

**while** ( CurrentPos >= TableSize ) CurrentPos -= TableSize**;** //实现取模

**}**

**else {**

CurrentPos = h0 - ( i -q ) \* ( i - q )**;**

**while** ( CurrentPos < 0 ) CurrentPos += TableSize**;** //实现取模

**}**

i++**;**

**}**

**if** ( ht[CurrentPos].info == Active **&&** ht[CurrentPos].Element == x )

**return** CurrentPos**;** //返回桶号

**else {**

ht[CurrentPos].info = Active**;**  ht[CurrentPos].Element = x**;** //插入x

**if** ( ++CurrentSize < TableSize / 2 ) **return** CurrentPos**;**

//当前已有项数加1, 不超过表长的一半返回

HashEntry \*Oldht = ht**;** //分裂空间处理**:** 保存原来的散列表

**int** OldTableSize = TableSize**;**

CurrentSize = 0**;**

TableSize = NextPrime ( 2 \* OldTableSize )**;** //原表大小的2倍，取质数

Allocateht ( )**;** //建立新的二倍大小的空表

**for** ( i = 0**;** i < OldTableSize**;** i++) //原来的元素重新散列到新表中

**if** ( Oldht[i].info == Active ) **{**

Find ( Oldht[i].Element )**;** //递归调用

**if** ( Oldht[i].Element == x ) CurrentPos = i**;**

**}**

**delete** [ ] Oldht**;**

**return** CurrentPos**;**

**}**

**}**

求下一个大于参数表中所给正整数*N*的质数的算法。

**int** NextPrime ( **int** N ) **{** //求下一个>N的质数，设N >= 5

**if** ( N % 2 == 0 ) N++**;** //偶数不是质数

**for** ( **;** !IsPrime (N)**;** N += 2 )**;** //寻找质数

**return** N**;**

**}**

**int** IsPrime ( **int** N ) **{** //测试N是否质数

**for** ( **int** i = 3**;** i\*i <= N**;** i += 2 ) //若N能分解为两个整数的乘积, 其中一个一定 

**if** ( N % i == 0 ) **return** 0**;** //N能整除i, N不是质数

**return** 1**;** //N是质数

**}**

10-16 编写一个算法，以字典顺序输出散列表中的所有标识符。设散列函数为hash(x) = x中的第一个字符，采用线性探查法来解决冲突。试估计该算法所需的时间。

【解答】

用线性探查法处理溢出时散列表的类的声明。

**#define** DefaultSize 1000

**#include** <iostream.h>

**#include** <string.h>

**#include** <stdlib.h>

**class** HashTable **{** //散列表类定义

**public:**

**enum** KindOfEntry **{** Active, Empty, Deleted **};** //表项分类 (活动 / 空 / 删)

HashTable ( ) **:** TableSize ( DefaultSize ) **{** ht = **new** HashEntry[TableSize]**;** **}** //构造函数

~HashTable ( ) **{ delete** [ ] ht**;** **}** //析构函数

**int** Find-Ins ( **const char** \* id )**;** //在散列表中搜索标识符id

**void** HashSort ( )**;**

**private:**

**struct** HashEntry **{** //表项定义

**Type** Element**;** //表项的数据, 即表项的关键码

KindOfEntry info**;** //三种状态: Active, Empty, Deleted

HashEntry ( ) **:** info (Empty ) **{ }** //表项构造函数, 置空

**};**

HashEntry \*ht**;** //散列表存储数组

**int** TableSize**;** //最大桶数

**int** FindPos ( **string** s) **const { return** atoi (\*s) - 32**; }** //散列函数

**}**

**int** HashTable **::** Find-Ins ( **const char** \* id ) **{**

**int** i = FindPos ( id )**,** j = i**;** //i是计算出来的散列地址

**while** ( ht[j].info != Empty **&&** strcmp ( ht[j].Element, id ) != 0 ) **{** //冲突

j = ( j + 1 ) % TableSize**;** //当做循环表处理, 找下一个空桶

**if** ( j == i ) **return** -TableSize**;** //转一圈回到开始点, 表已满, 失败

**}**

**if** ( **ht**[j].info != Active ) **{** //插入

**if** ( j > i ) **{**

**while** ( **int** k = j**;** k > i**;** k*--* )

**{** ht[k].Element = ht[k-1].Element**;**  ht[k].info = ht[k-1].info**; }**

ht[i].Element = id**;** ht[i].info = Active**;** //插入

**} else {**

HashEntry temp**;**

temp.Element = ht[TableSize-1].Element**;** temp.info = ht[TableSize-1].info**;**

**while** ( **int** k = TableSize-1**;** k > i**;** k*--* )

**{** ht[k].Element = ht[k-1].Element**;**  ht[k].info = ht[k-1].info**; }**

ht[i].Element = id**;** ht[i].info = Active**;** //插入

**while** ( **int** k = j**;** k > 0**;** k*--* )

**{** ht[k].Element = ht[k-1].Element**;**  ht[k].info = ht[k-1].info**; }**

ht[0].Element = temp.Element**;** ht[0].info = temp.info**;**

**}**

**return** j**;**

**}**

**void** HashTable **::** HashSort ( ) **{**

**int** n, i**; char** \* str**;**

**cin** >> n >> str**;**

**for** ( i = 0**;** i < n**;** i++ ) **{**

**if** ( Find-Ins ( str ) == - Tablesize ) **{ cout** << "表已满" << **endl; break; }**

**cin** >> str**;**

**}**

**for** ( i = 0**;** i < TableSize**;** i++ )

**if** ( ht[i].info == Active ) **cout** << ht[i].Element << **endl;**

**}**

10-17 设有1000个值在1到10000的整数，试设计一个利用散列方法的算法，以最少的数据比较次数和移动次数对它们进行排序。

【解答1】

建立TableSize = 10000的散列表，散列函数定义为

**int** HashTable **::** FindPos ( **const int** x ) **{ return** x-1**; }**

相应排序算法基于散列表类

**#define** DefaultSize 10000

**#define** n 1000

**class** HashTable **{** //散列表类的定义

**public:**

**enum** KindOfEntry **{** Active, Empty, Deleted **};** //表项分类 (活动 / 空 / 删)

HashTable ( ) **:** TableSize ( DefaultSize ) **{** ht = **new** HashEntry[TableSize ]**;** **}** //构造函数

~HashTable ( ) **{ delete** [ ] ht**;** **}** //析构函数

**void** HashSort ( **int** A[ ], **int** n )**;** //散列法排序

**private:**

**struct** HashEntry **{**  //散列表的表项定义

**int** Element**;** //表项的数据, 整数

KindOfEntry info**;** //三种状态: Active, Empty, Deleted

HashEntry ( ) **:** info (Empty ) **{ }** //表项构造函数

**};**

HashEntry \*ht**;** //散列表存储数组

**int** TableSize**;** //数组长度

**int** FindPos ( **int** x )**;** //散列函数

**};**

**void** HashTable**<Type> ::** HashSort ( **int** A[ ], **int** n ) **{** //散列法排序

**for ( int** i = 0**;** i < n**;** i++ **) {**

**int** position = FindPos( A[i] )**;**

ht[position].info = Active; ht[position].Element = A[i]**;**

**}**

**int** pos = 0**;**

**for ( int** i = 0**;** i < TableSize**;** i++ **)**

**if** ( ht[i].info *==* Active )

**{** **cout** << ht[i].Element << **endl;**  A[pos] = ht[i].Element**;** pos**++; }**

**}**

【解答2】

利用开散列的方法进行排序。其散列表类及散列表链结点类的定义如下：

**#define** DefaultSize 3334

**#define** n 1000

**class** HashTable**;** //散列表类的前视声明

**class**ListNode **{** //各桶中同义词子表的链结点(表项)定义

**friend** **class** HashTable**;**

**private:**

**int** key**;** //整数数据

ListNode \*link**;** //链指针

**public:**

ListNode ( **int** x ) **:** key(x), link(NULL) **{ }** //构造函数

**};**

**typedef** ListNode \*ListPtr**;** //链表指针

**class** HashTable **{** //散列表(表头指针向量)定义

**public:**

HashTable( **int** size = DefaultSize ) //散列表的构造函数

**{** TableSize = size**;** ht = **new** ListPtr[size]**; }** //确定容量及创建指针数组

**void** HashSort ( **int** A[ ]**; int** n )

**private:**

**int** TableSize**;** //容量(桶的个数)

ListPtr\*ht**;** //散列表定义

**int** FindPos ( **int** x ) **{ return** x / 3**; }**

**}**

**void** HashTable**<Type> ::** HashSort ( **int** A[ ]**; int** n ) **{**

ListPtr \* p , \*q**; int** i, bucket, k = 0**;**

**for** ( i = 0**;** i < n**;** i++ ) **{** //对所有数据散列, 同义词子表是有序链表

bucket = FindPos ( A[i] )**;** //通过一个散列函数计算桶号

p = ht[bucket]**;** q = **new** ListNode(A[i])**;**

**if** ( p == NULL || p->key > A[i] ) //空同义词子表或\*q的数据最小

**{** q->link = ht[bucked]**;** ht[bucked] = q**; }**

**else if** ( p->link == NULL || p->link->key > A[i] )

**{** q->link = p->link**;** p->link = q**; }**

**else** p->link->link = q**;** //同义词子表最多3个结点

**}**

**for** ( i = 0**;** i < TableSize**;** i++ ) **{** //按有序次序输出

p = ht[i]**;**

**while** ( p != NULL ) **{**

**cout** << p->key << **endl;**  A[k] = p->key**;**  k++**;**

p = p->link**;**

**}**

**}**

**}**

10-18 设有15000个记录需放在散列文件中，文件中每个桶内各页块采用链接方式连结，每个页块可存放30个记录。若采用按桶散列，且要求搜索到一个已有记录的平均读盘时间不超过1.5次，则该文件应设置多少个桶？

【解答】

已知用开散列法（开散列）解决冲突，搜索成功的平均搜索长度为1+α/2≤1.5，解出α≤1，又α= n / m = 15000 / 30 / m = 500 / m ≤1，m≥500。由此可知，该文件至少应设置500个桶。

10-19 用可扩充散列法组织文件时，若目录深度为d，指向某个页块的指针有n个，则该页块的局部深度有多大？

【解答】

设页块的局部深度为d'，根据题意有n = 2 d-d'，因此，d' = d - log2 n。

10-20 设一组对象的关键码为 **{** 69, 115, 110, 255, 185, 143, 208, 96, 63, 175, 160, 99, 171, 137, 149, 229, 167, 121, 204, 52, 127, 57, 1040 **}**。要求用散列函数将这些对象的关键码转换成二进制地址，存入用可扩充散列法组织的文件里。定义散列函数为hash(key) = key % 64, 二进制地址取6位。设每个页块可容纳4个对象。要求按10 .4节介绍的方法设置目录表的初始状态，使目录表的深度为3。然后按题中所给的顺序，将各个对象插入的可扩充散列文件中。试画出每次页块分裂或目录扩充时的状态和文件的最后状态。

【解答】

hash(69) = 5(10) = 000101(2) hash(115) = 51(10) = 110011(2)

hash(110) = 46(10) = 101110(2) hash(255) = 63(10) = 111111(2)

hash(185) = 57(10) = 111001(2) hash(143) = 15(10) = 001111(2)

hash(208) = 16(10) = 010000(2) hash(96) = 32(10) = 100000(2)

hash(63) = 63(10) = 111111(2) hash(175) = 47(10) = 101111(2)

hash(160) = 32(10) = 100000(2) hash(99) = 35(10) = 100011(2)

hash(171) = 43(10) = 101011(2) hash(137) = 9(10) = 001001(2)

hash(149) = 21(10) = 010101(2) hash(229) = 37(10) = 100101(2)

hash(167) = 39(10) = 101001(2) hash(121) = 57(10) = 111001(2)

hash(204) = 12(10) = 001100(2) hash(52) = 52(10) = 110100(2)

根据题意，每个页块可容纳4个对象，为画图清晰起见仅给出前20个关键码插入后的结果。目录表的深度d = 3。

DicDepth = 1

PgDepth = 1 Depth = 1

PgDepth = 1

DicDepth = 1

110

0

1

0

1

69, 115, 110, 255

PgDepth = 1

69, 115, 255, 185

插入185, 页块分裂

初态, 插入69, 115, 110, 255

DicDepth = 3

PgDepth = 1 Depth = 1

PgDepth = 1 Depth = 1

DicDepth = 2

000

001

010

011

100

101

110

111

69, 185, 208, 96

69, 185, 208, 96

00

01

10

11

PgDepth = 2

PgDepth = 2

110

PgDepth = 2

110

PgDepth = 3

115, 63

115, 255, 143, 63

PgDepth = 3

255, 143, 63, 175

插入143, 目录分裂, 插入208, 96, 63

插入175, 目录分裂

PgDepth = 2 Depth = 1

DicDepth = 3

PgDepth = 2 Depth = 1

DicDepth = 3

208, 96, 160, 204

000

001

010

011

100

101

110

111

208, 96, 160

000

001

010

011

100

101

110

111

PgDepth = 3 Depth = 1

PgDepth = 2 Depth = 1

185, 137, 167, 121

69, 185, 137, 149

PgDepth = 3 Depth = 1

PgDepth = 2

69, 149, 229

PgDepth = 2

110

PgDepth = 3

110

115, 63, 99, 171

PgDepth = 3

PgDepth = 3

115, 63, 99, 171

255, 143, 63, 175

PgDepth = 3

255, 143, 63, 175

插入160, 页块分裂, 插入99, 171, 137, 149

插入229, 页块分裂, 插入167, 121, 204

PgDepth = 3 Depth = 1

DicDepth = 3

208, 96, 160

000

001

010

011

100

101

110

111

PgDepth = 3 Depth = 1

204, 52

PgDepth = 3 Depth = 1

185, 137, 167, 121

PgDepth = 3 Depth = 1

69, 149, 229

PgDepth = 2

110

PgDepth = 3

115, 63, 99, 171

PgDepth = 3

255, 143, 63, 175

插入52, 页块分裂

**四、其他练习题**

10-21 判断下列各叙述的正误。正确的打“√”，错误的打“×”。

(1) 在索引顺序结构上实施分块搜索，在等概率情况下，其平均搜索长度不仅与子表个数有关，而且与每一个子表中的对象个数有关。

(2) 在索引顺序结构的搜索中，对索引表既可以采取顺序搜索，也可以采用折半搜索。

(3) 在9阶B树中，除根以外其他非叶结点中的关键码个数不少于4。

(4) 在9阶B树中，除根以外的任何一个非叶结点中的关键码个数均在5~9之间。

(5) 对于B树中任何一个非叶结点中的某个关键码k来说，比k大的最小关键码和比k小的最大关键码一定都在叶结点中。

(6) B树是一种动态索引结构, 它既适用于随机搜索, 也适用于顺序搜索。

(7) 理想情况下，在散列表中搜索一个元素的时间复杂度为O(1)。

(8) 在散列法中，一个可用的散列函数必须保证绝对不产生冲突。

(9) 在散列法中采取开散列（链地址）法来解决冲突时, 其装载因子的取值一定在(0,1)之间。

(10) 在散列法中采取闭散列（开地址）法来解决冲突时, 一般不要立刻做物理删除, 否则在搜索时会发生错误。

【解答】

(1) √ (2) √ (3) √ (4) × (5) √ (6) × (7) √ (8) ×

(9) × (10) √

10-22 单项选择题

[在备选答案中只有一个是正确的, 将其选出并把它的标号写在题后的括号内]

(1) 在10阶B树中根结点所包含的关键码个数最多为\_\_\_\_\_\_\_，最少为\_\_\_\_\_\_\_\_。

A. 1 B. 2 C. 9 D. 10

(2) 在一棵高度为h的B树中，叶结点处于第\_\_\_\_\_\_\_层，当向该B树中插入一个新关键码时，为搜索插入位置需读取\_\_\_\_\_\_\_\_个结点。

A. h-1 B. h C. h+1 D. h+2

(3) 当对一个线性表R[60] 进行索引顺序搜索（分块搜索）时，若共分成了10个子表，每个子表有6个表项。假定对索引表和数据子表都采用顺序搜索，则搜索每一个表项的平均搜索长度为\_\_\_\_\_\_\_\_。

A. 7 B. 8 C. 9 D. 10

(4). 既希望较快的搜索又便于线性表动态变化的搜索方法是\_\_\_\_\_\_\_\_。

A. 顺序搜索 B. 折半搜索 C. 散列搜索 D. 索引顺序搜索

(5) 散列函数有共同的性质，则函数值应当以\_\_\_\_\_\_\_\_概率取其值域的每一个值。

A. 最大 B. 最小

C. 平均 D. 同等

(6) 设散列地址空间为0 ~ m-1，k为表项的关键码，散列函数采用除留余数法，即Hash(k) = k % p。为了减少发生冲突的频率，一般取p为\_\_\_\_\_\_\_\_\_。

A. m B. 小于m的最大质数

C. 大于m的最小质数 D. 小于m的最大合数

(7) 在采用开散列法解决冲突时，每一个散列地址所链接的同义词子表中各个表项的\_\_\_\_\_\_\_\_相同。

A. 关键码值 B. 元素值

C. 散列地址 D. 含义

(8) 解决散列法中出现的冲突问题常采用的方法是\_\_\_\_\_\_\_\_。

A. 数字分析法、除留余数法、平方取中法

B. 数字分析法、除留余数法、线性探查法

C. 数字分析法、线性探查法、双散列法

D. 线性探查法、双散列法、开散列法

(9) 在闭散列表中，散列到同一个地址而引起的“堆积”问题是由于\_\_\_\_\_\_\_\_引起的。

A. 同义词之间发生冲突

B. 非同义词之间发生冲突

C. 同义词之间或非同义词之间发生冲突

D. 散列表“溢出”

(10) 采用线性探查法解决冲突时所产生的一系列后继散列地址\_\_\_\_\_\_\_。

A. 必须大于等于原散列地址

B. 必须小于等于原散列地址

C. 可以大于或小于但不等于原散列地址

D. 对地址在何处没有限制。

(11) 对包含n个元素的散列表进行搜索，平均搜索长度\_\_\_\_\_\_\_\_。

A. 为O(log2n) B、为O(n)

C. 不直接依赖于n D. 上述三者都不是

【解答】

(1) C, A (2) B, C (3) C (4) D (5) D (6) B

(7) A (8) D (9) B (10) C (11) C

10-23 填空题

(1) 在索引表中，每个索引项至少包含有\_\_\_\_\_\_\_\_域和\_\_\_\_\_\_\_\_域这两项。

(2) 假定一个线性表为 **{**12, 23, 74, 55, 63, 40, 82, 36 **}**，若按key % 3条件进行划分，使得同一余数的元素成为一个子表，则得到的三个子表分别为\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_。

(3) 假定一个线性表为(”abcd”,”baabd”,”bcef”,”cfg”,”ahij”,”bkwte”,”ccdt”,”aayb”)，若按照字符串的第一个字母进行划分，使得同一个字母被划分在一个子表中，则得到的a,b,c三个子表的长度分别为\_\_\_\_\_\_\_\_、\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_。

(4) 在索引表中，若一个索引项对应数据对象表中的一个表项，则称此索引为\_\_\_\_\_\_\_\_索引，若对应数据对象表中的若干表项，则称此索引为\_\_\_\_\_\_\_\_索引。

(5) 假定对长度n = 100的线性表进行索引顺序搜索，并假定每个子表的长度均为，则进行索引顺序搜索的平均搜索长度为\_\_\_\_\_\_\_\_，时间复杂度为\_\_\_\_\_\_\_\_。

(6) 若对长度n = 10000的线性表进行二级索引存储，每级索引表中的索引项是下一级20个表项的索引，则一级索引表的长度为\_\_\_\_\_\_\_\_，二级索引表的长度为\_\_\_\_\_\_\_\_。

(7) 假定要对长度n = 100的线性表进行散列存储，并采用开散列法处理冲突，则对于长度m = 20的散列表，每个散列地址的同义词子表（单链表）的长度平均为\_\_\_\_\_\_\_\_。

(8) 在线性表的散列存储中，装载因子 α 又称为装载系数，若用m表示散列表的长度，n表示待散列存储的元素的个数，则 α 等于\_\_\_\_\_\_\_\_。

(9) 对于包含n个关键码的m阶B树，其最小高度为\_\_\_\_\_\_\_\_，最大高度为\_\_\_\_\_\_\_\_。

(10) 已知一棵3阶B树中含有50个关键码，则该树的最小高度为\_\_\_\_\_\_\_\_，最大高度为\_\_\_\_\_\_\_\_。

(11) 在一棵m阶B树上，每个非根结点的关键码数最少为\_\_\_\_\_\_\_\_个，最多为\_\_\_\_\_\_\_\_个，其子树棵数最少为\_\_\_\_\_\_\_\_，最多为\_\_\_\_\_\_\_\_。

(12) 在一棵B树中，所有叶结点都处在\_\_\_\_\_\_\_\_上，所有叶结点中空指针等于所有\_\_\_\_\_\_\_\_总数加1。

(13) 在对m阶B树插入元素的过程中，每向一个结点插入一个关键码后，若该结点的关键码个数等于\_\_\_\_\_\_\_个，则必须把它分裂为\_\_\_\_\_\_\_\_个结点。

(14) 在从m阶B树删除关键码的过程中，当从一个结点中删除掉一个关键码后，所含关键码个数等于\_\_\_\_\_\_\_\_个，并且它的左、右兄弟结点中的关键码个数均等于\_\_\_\_\_\_\_\_，则必须进行结点合并。

(15) 向一棵B\_树插入关键码的过程中，若最终引起树根结点的分裂，则新树比原树的高度\_\_\_\_\_\_\_\_。

(16) 从一棵B树删除关键码的过程中，若最终引起树根结点的合并，则新树比原树的高度\_\_\_\_\_\_\_\_。

【解答】

(1) 关键码值, 子表地址域 (2) (12, 63, 36), (55, 40, 82), (23, 74)

(3) 3, 3, 2 (4) 稠密, 稀疏

(5) 11, O() (6) 500, 25

(7) 5 (8) n/m

(9) ⎡logm(n+1)⎤ , ⎣log⎡m/2⎤ ( (n+1)/2)⎦ (10) 4, 5

(11) ⎡m/2⎤-1, m-1, ⎡m/2⎤ , m (12) 同一层, 关键码

(13) m, 2 (14) ⎡m/2⎤-2, ⎡m/2⎤-1

(15) 增1 (16) 减1

10-24请回答有关二次散列方的问题

若用二次散列解决冲突，求下一个“空位”的探查序列为

Hi = ( H0 + i2 ) % m, Hi = ( H0 - i2 ) % m, i = 1, 2, …, m/2.

其中，H0是第一次求得的散列地址，Hi是第i次求得的散列地址，m是散列表的大小。

(1) 相邻的地址Hi与Hi-1之间是什么关系？

(2) 为保证散列地址序列的地址不会循而往复地重迭，m应设为什么数？装载因子α的取法如何？

(3) 为保证在删除时不中断查找链，可对被删记录做逻辑删除。为此，每个散列表的表项有3个状态，除了Active（正在使用）和Deleted（删除）状态外，还应有一个什么状态？

【解答】

(1) , i = 1, 2, …, m/2

(2) 表的大小m应为满足4k+3的整数，其中k是正整数。装载因子 α≤0.5。

(3) 散列表中每个表项有3个状态：Active（正在使用）、Deleted（删除）、Empty（空）。

10-25假定有一个100×100的稀疏矩阵，其中1%的元素为非零元素，现要求对其非零元素进行散列存储，使之能够按照元素的行、列值存取矩阵元素（即元素的行、列、值联合为元素的关键字），试采用除留余数法构造散列函数和线性探查法处理冲突，分别写出建立散列表和查找散列表的算法。

【解答】

由题意可知，整个稀疏矩阵中非零元素的个数为100。为了散列存储这100个非零元素，需要使用一个作为散列表的一维数组，该数组中元素的类型应为： **struct** ElemType **{**

**int** row**;** //存储非零元素的行下标

**int** col**;** //存储非零元素的列下标

**float** val**;** //存储非零元素值

**};**

假定用HT[m]表示这个散列表，其中m为散列表的长度，若取装载因子为0.8左右，则令m为127为宜（因127为质数）。

按照题目要求，需根据稀疏矩阵元素的行下标和列下标存取散列表中的元素，所以每个元素的行下标和列下标同时为元素的关键字。假定用x表示一个非零元素，按除留余数法构造散列函数，并考虑尽量让得到的散列地址分布均匀，所以采用的散列函数为：

Hash(x) = (13 \* x.row + 17 \* x.col ) % m

根据以上分析，建立散列表的算法如下：

**int** HashTable <ElemType> **: :** Create ( ) **{**

//根据稀疏矩阵中100个非零元素建立散列表

**int** i, d, temp**;** ElemType x**;**

**for** ( i = 0**;** i < TableSize**;** i++ ) //散列表初始化, 行、列号码置为 –1, 值置为0

**{** ht[i].row = -1**;** ht[i].col = -1**;** ht[i].val = 0**; }**

**for** ( i = 1**;** i <= 100**;** i++ ) **{** //循环, 输入一个非零元素并插入到散列表中

**cout** << i << "**:** "**;**

**cin** >> x.row >> x.col >> x.val**;** //输入非零元素

d = (13 \* x.row + 17 \* x.col ) % TableSize**;** //计算初始散列地址

temp = d**;**

**while** ( ht[d].val != 0 ) **{** //线性探查存储位置

//此循环条件也可用ht[d].row! = -1或ht[d].col != -1来代替

d = (d+1) % TableSize**;**

**if** ( d == temp ) **return** 0**;** //无插入位置返回0

**}**

ht[d] = x**;**  //非零元素存入 下标d位置

**}**

**return** 1**;** //全部元素插入成功后返回1

**}**

在散列表上进行查找的算法如下：

**int** HashTable <ElemType> **: :** Search ( **int** row**, int** col ) {

//采用与插入时使用的同一散列函数计算散列地址

**int** d = ( 13\*row + 17\*col ) % TableSize**;**

//采用线性探查法查找行、列下标分别为row和col的元素

**while** ( ht[d].val != 0 ) **{**

//此循环条件也可用ht[d].row != -1或ht[d].col != -1来代替

**if** ( ht[d].row == row **&&** ht[d].col == col )

**return** d**;** //查找成功返回元素的下标

**else** d = (d +1) % TableSize**;**

**if** ( d == temp ) **return** -1**;**

**}**

**return** -1**;** //查找失败返回-1

**}**