常见字符串哈希函数及冲突处理方法的查找性能评估

中山大学电子与信息工程学院

15 通信工程　冼宇乔

摘要

哈希表（Hash Table，也称散列表）是一种通过哈希函数将关键码值映射到哈希表地址而直接进行访问的数据结构，用以加快查找的速度。理想的哈希函数应能将关键字随机、均匀地映射到哈希表的不同地址，并且不产生冲突（不同的关键字不会有相同的哈希地址）。但是由于哈希函数的不可预测性，无法保证冲突不产生，因此必须使用一定的冲突处理方法来处理出现的哈希地址冲突，常见的方法有开放定址法、拉链法等。本文将以平均探测次数为指标，采用 CET-4 单词库作为储存/查找对象，编写程序考察常见的 8 种字符串哈希函数及 4 种哈希冲突处理方法在不同的负载因子（关键字数目/哈希表地址范围）下的性能。

1. 目标

现有一个 CET-4 英语单词及其释义的文本，使用哈希方法储存其数据，实验程序能够读入输入文件，输入文件中包含若干个需要查询的英语单词，程序将在 CET-4 单词库中查找这些单词是否存在，如果存在就输出它们的中文释义。采用不同的哈希函数与哈希表冲突处理方法，统计其在不同负载因子下的平均探测次数以评估其性能以及优缺点。

2. 哈希方法

实验程序实现了 8 种常见的字符串哈希函数，使用统一的函数接口，封装在头文件“hash.h”中。在主程序中使用函数指针来调用这八个哈希函数。

这八个哈希函数分别为 ELF Hash, SDBM Hash, RS Hash, JS Hash, PJW Hash, BKDR Hash, DJB Hash, AP Hash。这些哈希大多使用了位运算的方法，使得字符串里的每一个字符都对哈希值产生影响。这些哈希函数的原理和具体实现请阅读参考文献与头文件代码，在本文将不予叙述。

3. 冲突处理方法

实验程序实现了 4 种常见的哈希表冲突处理方法，包括属于开放定址法的线性探测和二次探测，以及属于基于单链表储存结构拉链法和基于二叉排序树（红黑树）储存结构的拉链法。

对于开放定址法，采用 STL 中的 vector 来存放哈希表元素；对于基于单链表储存结构拉链法，使用 STL 中的 list 来存放哈希表元素；对于基于二叉排序树（红黑树）储存结构拉链法，使用 STL 中的 set 来存放哈希表元素（因为 set 内部使用红黑树实现，集合里的元素内部非递减有序）。

线性探测法是形式最简单的处理冲突的方法。插入元素时，如果发生冲突，算法会简单的从该位置向后循环遍历哈希表，直到找到表中的下一个空的位置，并将该元素放入该槽中。查找元素时，首先散列值所指向的槽，如果没有找到匹配，则继续从该槽遍历哈希表，直到：（1）找到相应的元素；（2）找到一个空槽，指示查找的元素不存在；（3）整个 hash 表遍历完毕（指示该元素不存在并且 hash 表是满的）

用线性探测法处理冲突，思路清晰，算法简单，但存在下列缺点：

①哈希表负载因子不能大于 1，如果要储存的关键字数目大于哈希表地址长度，则会发生溢出，必须重新申请一片新的地址空间来存放数据。

②删除工作非常困难。如果将表中某一元素直接删除，查找的时会发现空槽，则会认为要找的元素不存在。只能标上已被删除的标记，否则，将会影响以后的查找。

③线性探测法很容易产生堆聚现象。所谓堆聚现象，就是存入哈希表的记录在表中连成一片。按照线性探测法处理冲突，如果生成哈希地址的连续序列愈长（即不同关键字值的哈希地址相邻在一起愈长），则当新的记录加入该表时，与这个序列发生冲突的可能性愈大。因此，哈希地址的较长连续序列比较短连续序列生长得快，这就意味着，一旦出现堆聚（伴随着冲突），就将引起进一步的堆聚。

二次探测法是对线性探测法的改进，它将线性探测处理冲突时向后探测的步长从 1 改为 n^2，降低了进一步发生冲突和堆聚的几率，但仍然存在和线性探测一样的缺点。

拉链法处理冲突的方式是，插入元素时，如果表中对应哈希值的位置已经有元素，则使用链式结构（链表或二叉树）将要插入的元素与已存在的元素链接起来。查找元素时，如果表中对应哈希值的位置的元素不是要查找的元素，则顺着链式结构继续查找，直到查找成功或查找失败为止。如果使用链表结构，则插入和查找元素的复杂度都是 O（n），而使用二叉排序树（红黑树）进行优化，可以将插入和查找的复杂度降低到 O（logn），减少查找时间。但由于使用二叉排序树结构需要比链表更多的额外空间，在负载因子α较小，冲突较少时，使用链表结构会更节省空间。

与开放定址法相比，拉链法有如下几个优点：

①拉链法处理冲突简单，且无堆积现象，即非同义词决不会发生冲突，因此平均查找长度较短；

②由于拉链法中各链表上的结点空间是动态申请的，故它更适合于造表前无法确定表长的情况；

③开放定址法为减少冲突，要求负载因子α较小，故当结点规模较大时会浪费很多空间。而拉链法中可取α≥1，且结点较大时，拉链法中增加的指针域可忽略不计，因此节省空间；

④在用拉链法构造的散列表中，删除结点的操作易于实现。只要简单地删去链表上相应的结点即可。

拉链法的缺点是：指针需要额外的空间，故当结点规模较小时，开放定址法较为节省空间，而若将节省的指针空间用来扩大散列表的规模，可使装填因子变小，这又减少了开放定址法中的冲突，从而提高平均查找速度。

4. 测试数据生成方法

测试数据为若干个要查询的单词，如果要查询的单词是单词表里的单词，则为查找成功，否则为查找失败。测试数据中查找成功的单词所占比例被称为查找成功率（Success Rate）。

实验程序中的 void TestSampleGenerate(int SampleSize, double SuccessRate, string &FileName) 函数用于自动生成随机测试样本，用户可自定义测试样本大小（SampleSize），测试样本查找成功率（SuccessRate）以及生成的输入文件的文件名（FileName）。

测试数据的生成实现方法：（1）查找成功的单词生成方法：使用 C 语言库中的随机数生成函数 rand()，产生 0~3208 之间的随机数，再在存放单词表单词的数组中取出对应编号的单词输出到生成的文件中。（2）查找失败的单词生成方法：产生 1~9 之间的随机数作为随机单词的长度，然后随机产生一个该长度的英文单词，在哈希表中查找该随机生成的单词，若查找失败，则将此单词输出到生成的文件中，否则该单词不符合条件，重新生成另一个单词。

5. 程序设计

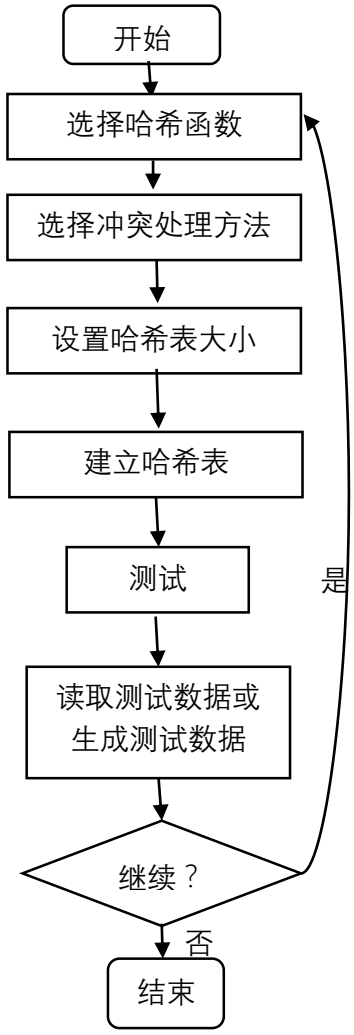

图 1：程序设计流程图

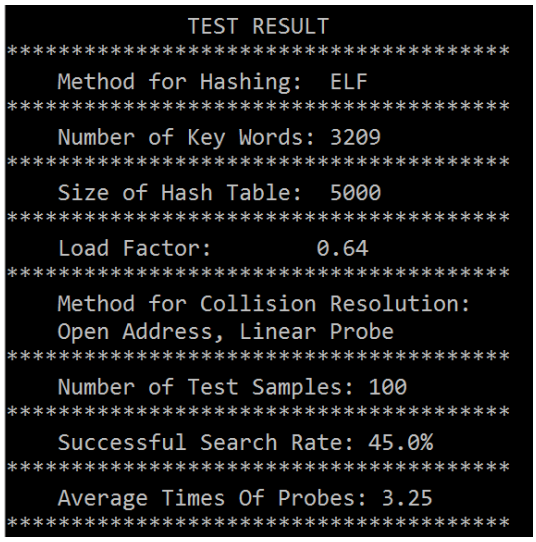

图 2：样本测试结果输出样例图

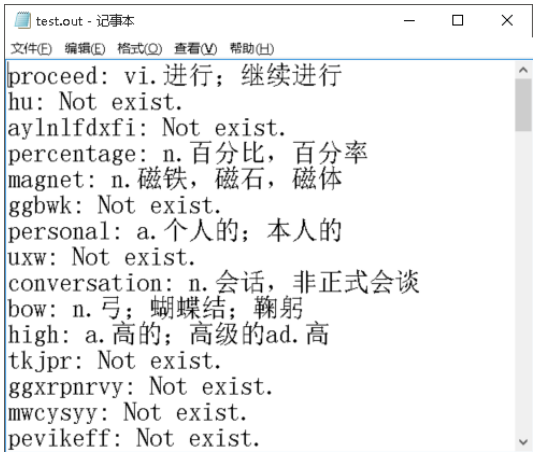

图 3：输出文件样例图

6. 测试

（1）哈希函数的性能评估：

对 8 种哈希函数分别在四种情况下进行测试，分别为①负载因子 0.5，线性探测；②负载因子 0.8，线性探测；③负载因子 0.99，链表拉链法；④负载因子 2，链表拉链法。测试样本为随机生成的样本大小为 20000，成功率为 0.5 的随机样本。分别统计四种情况下的平均探测次数，并计算其平均值。平均值越小，证明相同情况下使用该哈希函数的平均探测次数越少，该哈希函数的性能越好。测试数据如下：

函数	①	②	③	④	平均
ELF	2.03	18.57	1.78	3	6.345
SDBM	1.96	7.85	1.72	2.52	3.513
RS	2.08	9.26	1.77	2.46	3.893
JS	1.93	8.46	1.73	2.53	3.663
PJW	2.03	18.57	1.78	2.53	6.228
BKDR	1.95	8.04	1.79	2.47	3.563
DJB	2.09	13.55	1.73	2.5	4.968
AP	2.04	8.78	1.77	2.51	3.775

表 1 ： 8 种哈希函数的平均探测次数

（2）哈希表冲突处理方法的性能评估：

采用实验程序的自动生成随机测试样本功能，生成两个样本大小为 10000，查找成功率分别为 1 和 0 的两个测试样本，分别对应查找成功和查找失败，并测试 4 种冲突处理方法在不同负载因子下的平均探测次数。测试结果如下：

负载因子	①	②	③	④
0.02	1.01	1.01	1.01	1.03
0.1	1.06	1.06	1.05	1.1
0.3	1.23	1.21	1.15	1.27
0.5	1.52	1.44	1.23	1.38
0.8	6.48	2.43	1.38	1.53
0.99	25.51	4.71	1.49	1.63
2	-	-	2.03	1.92
5	-	-	3.42	2.53

表 2：4 种冲突处理方法的平均探测次数（查找成功）

负载因子	①	②	③	④
0.02	1.02	1.02	1.02	1
0.1	1.13	1.14	1.1	1
0.3	1.57	1.62	1.3	1.04
0.5	2.64	2.47	1.49	1.1
0.8	36.56	7.48	1.78	1.22
0.99	1121	81.21	1.97	1.32
2	-	-	3	1.91
5	-	-	5.97	3.21

表 3：4 种冲突处理方法的平均探测次数（查找失败）

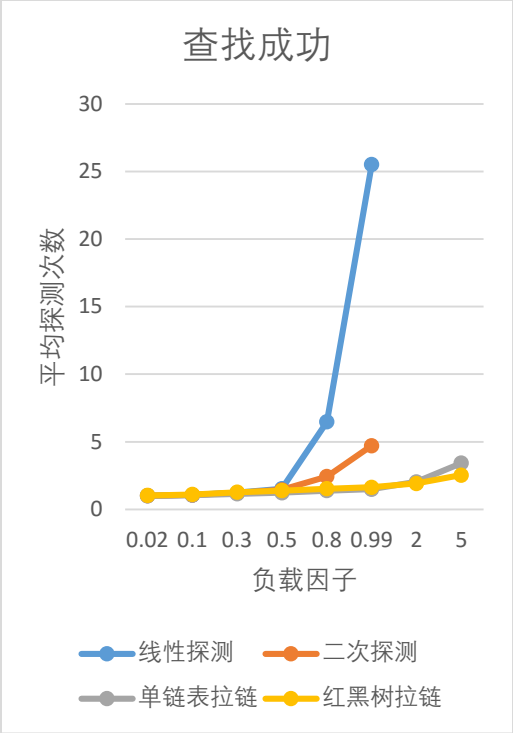

图 4：4 种冲突处理方法在查找成功时的平均探测次数

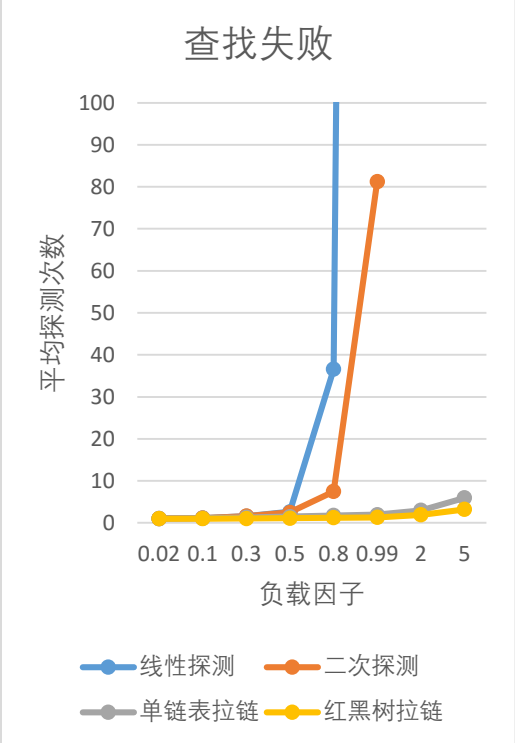

图 5：4 种冲突处理方法在查找失败时的平均探测次数

7. 结论

哈希表是一种典型的蕴含 “用空间换时间” 算法思想的数据结构，哈希函数的目标是实现从关键字到地址的平均映射，完美的哈希函数应该是不产生冲突的哈希函数。而不同的冲突处理方法则是对空间与时间消耗进行权衡。

哈希函数性能评估：这八种哈希函数性能良好，一般情况下能将字符关键字比较随机，均匀的分布到哈希表中。在情况②（负载因子 0.8，线性探测），ELF、PJW 和 DJB 三种函数的平均探测次数均超过了 10，这很有可能是线性探测方式造成的堆积而产生的探测次数增加。在情况③与④采用拉链法的情况下，这些哈希函数即使在负载因子很大的情况仍能保持很好的探测性能。对四种情况进行平均统计，得到 8 种哈希函数在测试样本中 表现出的性能从好到坏排序，分别是：（1）SDBM（3.513）；（2）BKDR（3.563）；（3）JS（3.663）；（4）AP（3.775）；（5）RS（3.893）；（6）DJB（4.968）；（7）PJW（6.228）；（8）ELF（6.345）。由于探测次数受负载因子和冲突处理方法的影响较大，以及随机样本具有一定的不确定性，所以这个排序只能提供一定的参考价值，并不能确定这八种哈希函数的优劣性能。在实际使用中，这八种哈希函数的性能一般情况下没有明显的差距。

冲突处理方法评估：一般情况下，查找失败时需要的平均探测次数大于查找成功的平均探测次数。在负载因子小于 0.5 时，由于哈希函数的性能较好，关键字分布比较平均，冲突较少，四种处理方法没有明显差距，平均探测次数均小于 2。当负载因子大于 0.5 时，开放地址法的平均探测次数呈指数式增长，其中线性探测增长得比二次探测更快。在查找失败，负载因子 0.99 这种极端情况下，线性探测平均次数达到了 1121 次，与哈希表大小为同一数量级。而拉链法呈线性增长，即使在负载因子达到 5 时，查找性能仍然优良，只需要线性的时间复杂度，而基于红黑树结构的拉链法性能略优与基于拉链法的性能。因此在实际使用中，应该根据哈希表的负载因子来选择冲突处理方法，如果负载因子小于 0.5，应该使用开放定址法，因为开放定址法不要要申请额外空间，而节省的空间可以用来进一步增加哈希表长度，减小负载因子。如果负载因子大于 0.5 应该使用拉链法。单链表拉链法相对红黑树拉链法更省空间，但需要更多的探测次数，应该视具体情况而进行权衡。

【参考文献】

【1】Robert L. Kruse, A J. Ryba, Data Structures And Program Design in C++, 2001;

【2】Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms, Third Edition;

【3】Anany Levitin, Introduction to the Design and Analysis of Algorithms, Third Edition;

【4】Robert Sedgewick, Kevin Wayn, Algorithms, Fourth Edition;