完美哈希技术

1）调研完美哈希的概念、特点。
2）针对业界使用完美哈希大多数针对静态数据的情况，调研动态完美哈希的实现策略、算法、特性。
3）讲述完美哈希的插入、检索效率，讲述完美哈希的空间占用特性；
4）与其他哈希的差别之处。

概述

定义

哈希表是一个根据 key 值直接访问数据的数据结构，记录了 key 和存储地址之间的映射关系，**哈希函数**是 key 值映射成地址的函数

完美哈希函数（Perfect Hash Function，简称 PHF）是没指有冲突的哈希函数。函数 H 将 N 个 key 值映射到 M 个整数上，这里 M>=N，且满足

$H(key1) \neq H(key2) \quad \forall key1, key2$，则该函数是完美哈希函数。

如果 M=N，则 H 是**最小完美哈希函数**（Minimal Perfect Hash Function，简称 MPHF）。此时 N 个 key 值会被映射到 N 个连续的整数上。MPHF 广泛用于从静态集合中快速存储和检索项目，例如自然语言中的单词、编程语言的保留词、Web 搜索引擎中的 URL 等。

对一个最小完美哈希函数，若 key 以$a_1, a_2, \ldots, a_n$给出，对于$\forall a_i, a_j \ i < j < n$，都满足$H(a_i) < H(a_j)$，则该最小完美哈希函数是**保序/单调**的。

如果一个哈希函数在给定的区域中有不超过 t 次冲突，那么这个哈希函数叫 **t-完美哈希函数**

负载因子 $\alpha = \frac{\text{表中的元素个数}}{\text{哈希表的总长度}}$，一般负载因子 α 越大则产生冲突的可能性越大。

对比

优点：相对于普通哈希，不会出现冲突，所以查询的时间复杂度为$O(1)$，优秀的 PHF 算法可以做到在线性时间$O(n)$构建一个 PHF

缺点：完美哈希函数就意味着事前必须知道需要哈希哪些数据。同时生成的算法比较复杂，需要很长的时间来建立索引。

名称	普通哈希	完美哈希
冲突	可能产生	通过构造特定哈希函数，不会产生冲突
数据集	未知	已知
查询效率	理想情况是 O(1)	稳定在 O(1)
插入新数据	如果产生冲突，则需要通过开放定址法或链接法等方法再次寻址	如果产生冲突，则需要重新选择哈希函数构建哈希表

算法

原理概述

关于生成 PHF 和 MPHF 有非常多原理不同的算法，有基于两级哈希表或者随机非循环图等

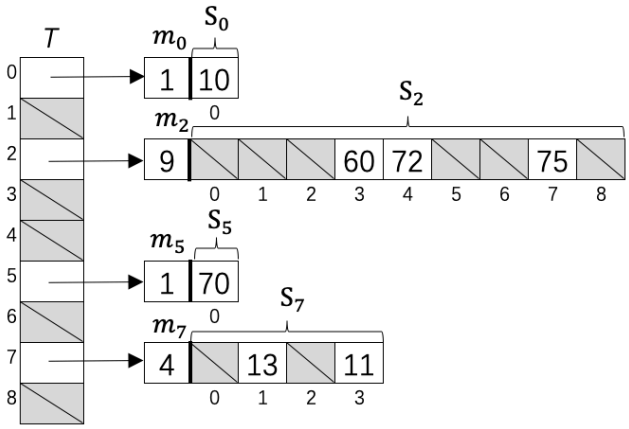

图 1

上图是两级哈希的 FKS 策略[1]，首先将数据通过第一级哈希映射到 T 空间中，然后冲突的数据随机选取新的哈希函数映射到 S 空间中，且 S 空间的大小 m 是冲突数据的平方（例如 T2 中有三个数字产生冲突，则映射到 m 为 9 的 S2 空间中），此时可以容易找到避免碰撞的哈希函数。最差情况下所需存储空间为$O(n^2)$，但只要适当选择哈希函数减少一级哈希时的碰撞，则可以使预期存储空间为$O(n)$。

动态完美哈希函数[2][3]也是近似于上述 FKS 策略，因为二级哈比表比较稀疏（负载因子低），所以在插入数据时出现冲突的概率较低，但出现冲突时需要随机选择新的哈希函数对二级哈希表进行重建。而对于其他负载因子更高的 PHF 或 MPHF 算法，插入数据可能要对哈希表进行**完全重建**。

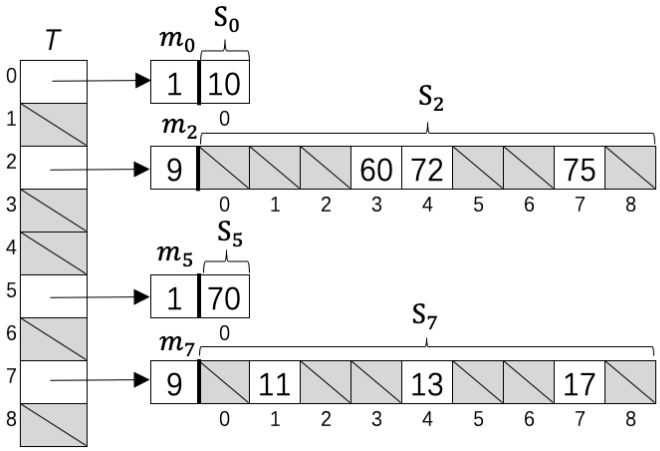

图 2

MPHF 的构建方法通常是叫 MOS(mapping, ordering, searching)。第一步 mapping 是将原始的 key 集合映射成换成一个新的集合， 第二步 ordering 将 key 值排成有序的队列，决定 key 和 hash 值的对应关系，第三步 searching 尝试将 hash 值分配给 key。

如图 2 所示，基于图 1 插入新的数据 17，如果落入 T 空间索引 7 的位置且发生冲突，则调整m_7为 9，并重新选择哈希函数重建S_7空间。

而对于其他负载因子更高的 PHF 或 MPHF 算法，插入数据时可能要对哈希表进行**完全重建**。

静态完美哈希算法

名称	特点	保序	key 存储占用
CHD[4]	1.构建 PHF 和 MPHF 的最快算法，线性时间 2.生成最紧凑的 PHF 和 MPHF 3.可以用于生成 t-完美哈希函数 4.基于两级哈希构建	未知	MPHF:2.07bits/key PHF:1.4bits/key
BDZ[5]	1.线性时间构建 PHF 和 MPHF 2.基于随机非循环图，其中每条边连接 3 个点	否	PHF:1.95 bits/key MPHF:(2+x)cn bits(c≥1.23,x＞0) /key
BMZ[6][7]	1.线性时间构建 MPHF，比 CHM 快	否	MPHF:4cn bits/key

	2.基于随机非循环图 3.比 CHM 更紧凑		(0.93≤c≤1.15)
BRZ[8]	1.基于外部存储器的算法，可以为 10 亿数据构建 MPHF	否	<8bits/key
CHM[9]	1.线性时间构建 MPHF 2.基于随机非循环图	是	MPHF:4cn bits/key (c>2)
FCH[10]	1.只适用于较小的数据集 2.属于 BRZ 算法的一部分	未知	<4bit/key

动态完美哈希算法

早先有 FKS、Cuckoo[11]和对两种使用 Bloom Filter 进行拓展[12][13]的动态完美哈希算法，但对于每个 key 都需要较大的存储占用，且负载因子较低

Derek Pao 等人[14]在 2013 年提出了一个基于 bit-shuffle 和 bit-extraction 动态完美哈希算法，可玩以实现 80%-100%的负载因子，对于 100 万个数据，可以在几秒钟的 CPU 时间内（Windows XP，Intel Core2 6400 CPU @2.13GHz）构造出来，可用于高速查表的一些实时应用如 IP 地址查找、数据包分类，模式匹配和命名实时网络。

名称	存储占用	负载因子
FKS	未知	20%-30%
Cuckoo	40 bits/key	<50%
Extended BF1	40 bits/key	25%-30%
Extended BF2	50 bits/key	25%
Derek Pao Method	7-15 bits/key	80%-100%

常见生成器

gperf[15]

一个使用 C++编写的 GNU 完美哈希函数生成器

CMPH[16]

采用 C 代码编写，可以工作在 Linux 和 Win32 系统上，为超过 1 亿个数据的集合生成 MPHF，支持 CHD、BDZ、BMZ、BRZ、CHM、FCH 算法

Rust-PHF[17]

基于 Rust 编写编写的 PHF 生成器，使用 CHD 算法在 0.4 秒内为 10 万个数据的集合生成 PHF

参考文献

[1] Fredman, M. L., Komlós, J., and Szemerédi, E. 1984. Storing a Sparse Table with 0(1) Worst Case Access Time. J. ACM 31, 3 (Jun. 1984), 538-544

[2] Dietzfelbinger, M., Karlin, A., Mehlhorn, K., Meyer auf der Heide, F., Rohnert, H., and Tarjan, R. E. 1994. "Dynamic Perfect Hashing: Upper and Lower Bounds" Archived 2016-03-04 at the Wayback Machine. SIAM J. Comput. 23, 4 (Aug. 1994), 738-761

[3] Lu Y, Prabhakar B, Bonomi F. Perfect hashing for network applications[C]//2006 IEEE International Symposium on Information Theory. IEEE, 2006: 2774-2778.

[4] Belazzougui D, Botelho F C, Dietzfelbinger M. Hash, displace, and compress[C]//European Symposium on Algorithms. Springer, Berlin, Heidelberg, 2009: 682-693.

[5] F. C. Botelho, R. Pagh, N. Ziviani. Simple and space-efficient minimal perfect hash functions. In Proceedings of the 10th International Workshop on Algorithms and Data Structures (WADs'07), Springer-Verlag Lecture Notes in Computer Science, vol. 4619, Halifax, Canada, August 2007, 139-150.

[6] F. C. Botelho, D. Menoti, N. Ziviani. A New algorithm for constructing minimal perfect hash functions, Technical Report TR004/04, Department of Computer Science, Federal University of Minas Gerais, 2004.

[7] F. C. Botelho, Y. Kohayakawa, and N. Ziviani. A Practical Minimal Perfect Hashing Method. 4th International Workshop on efficient and Experimental Algorithms (WEA05), Springer-Verlag Lecture Notes in Computer Science, vol. 3505, Santorini Island, Greece, May 2005, 488-500.

[8] F. C. Botelho, Y. Kohayakawa, N. Ziviani. An Approach for Minimal Perfect Hash Functions for Very Large Databases, Technical Report TR003/06, Department of Computer Science, Federal University of Minas Gerais, 2004.

[9] Z.J. Czech, G. Havas, and B.S. Majewski. An optimal algorithm for generating minimal perfect hash functions. Information Processing Letters, 43(5):257–264,1992.

[10] E.A. Fox, Q.F. Chen, and L.S. Heath. A faster algorithm for constructing minimal perfect hash functions. In Proc. 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 266-273, 1992.

[11] R. Pagh and F. F. Rodler, “Cuckoo Hashing”, J. of Algorithm, Vol. 51, No. 2, pp. 122-144, 2004.

[12] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, “Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing”, ACM SIGCOMM, pp. 181-192, 2005.

[13] Y. Lu, B. Prabhakar, and F. Bonomi, “Perfect Hashing for Network Applications”, IEEE Symp. on Information Theory, pp. 2774-2778, 2006.

[14] Pao D, Wang X, Lu Z. Design of a near-minimal dynamic perfect hash function on embedded device[C]//2013 15th International Conference on Advanced Communications Technology (ICACT). IEEE, 2013: 457-462.

[15] http://www.gnu.org/software/gperf/

[16] http://cmph.sourceforge.net/index.html

[17] https://github.com/sfackler/rust-phf