

Extendible Hashing

Donghui Zhang
College of Computer & Information Science
Northeastern University
Boston, MA, USA
donghui@ccs.neu.edu

Yannis Manolopoulos
Department of Informatics
Aristotle University
Thessaloniki, 54124 Greece
manolopo@csd.auth.gr

Yannis Theodoridis
Department of Informatics
University of Piraeus
18534 Piraeus, Greece
ytheod@unipi.gr

Vassilis J. Tsotras
Department of Computer Science
and Engineering
University of California - Riverside
Riverside, CA 92521
tsotras@cs.ucr.edu

DEFINITION

Extendible Hashing is a dynamically updateable disk-based index structure which implements a hashing scheme utilizing a directory. The index is used to support exact match queries, i.e. find the record with a given key. Compared with the B+-tree index which also supports exact match queries (in logarithmic number of I/Os), Extendible Hashing has better expected query cost $O(1)$ I/O. Compared with Linear Hashing, Extendible Hashing does not have any overflow page. Overflows are handled by doubling the directory which logically doubles the number of buckets. Physically, only the overflowed bucket is split.

HISTORICAL BACKGROUND

A hash table is an in-memory data structure that associates keys with values. The primary operation it supports efficiently is a lookup: given a key, find the corresponding value. It works by transforming the key using a hash function into a hash, a number that is used as an index in an array to locate the desired location where the values should be. Multiple keys may be hashed to the same bucket, and all keys in a bucket should be searched upon a query. Hash tables are often used to implement associative arrays, sets and caches. Like arrays, hash tables have $O(1)$ lookup cost on average.

SCIENTIFIC FUNDAMENTALS

The Extendible Hashing scheme was introduced by [1].

Structure Extendible Hashing uses a directory to access its buckets. This directory is usually small enough to be kept in main memory and has the form of an array with 2^d entries, each entry storing a bucket address (pointer to a bucket). The variable d is called the global depth of the directory. To decide where a key k is stored, Extendible Hashing uses the last d bits of some adopted hash function $h(k)$ to choose the directory entry. Multiple directory entries may point to the same bucket. Every bucket has a local depth $leq d$. The difference between local depth and global depth affects overflow handling.

An example of Extendible Hashing is shown in Figure 1. Here there are four directory entries and four buckets. The global depth and all the four local depths are all 2. For simplicity assume the adopted hash function is $h(k) = k$. For instance, to search for record 15, one refers to directory entry $15 \% 4 = 3$ (or 11 in binary format), which points to bucket D .

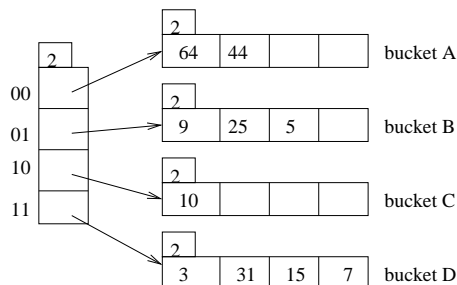


Figure 1: Illustration of the Extendible Hashing.

Overflow Handling If a bucket overflow happens, the bucket is split into two. The directory may or may not double, depending on whether the local depth of the overflow bucket was equal to the global depth before split.

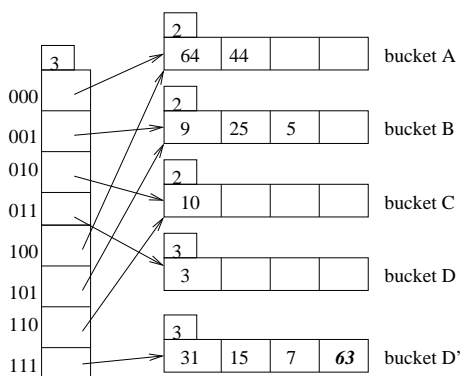


Figure 2: The directory doubles after inserting 63 into Figure 1.

If the local depth was equal to global depth, d bits are not enough to distinguish the search values of the overflow bucket. Thus a directory *doubling* occurs, which effectively uses one more bit from the hash value. The directory size is then doubled (this does not mean that the number of buckets is doubled as buckets will share directory entries). As an example, Figure 2 illustrates the Extendible Hashing after inserting a new record with key 63 into Figure 1. Bucket D overflows and the records in it are redistributed between D (where the last three bits of a record's hash value are 011) and D' (where the last three bits of a record's hash value are 111). The directory doubles. The global depth is increased by 1. The local depth of buckets D and D' are increased by 1, while the local depth of the other buckets remains to be 2. Except 111 which points to the new bucket D' , each of the new directory entries points to the existing bucket which shares the last 2 bits. For instance, directory entry 101 points to the bucket referenced by directory entry 001.

In general, let the local depth of a bucket be d' , the number of directory entries pointing to the bucket is $2^{d-d'}$. All these directory entries share the last d' bits.

To split an overflow bucket whose local depth is smaller than the global depth, one does not need to double the size of the directory. Instead, half of the $2^{d-d'}$ directory entries will point to the new bucket, and the local depth of both the overflow bucket and its split image are increased by 1. For instance, Figure 3 illustrates the Extendible Hashing after inserting 17 and 13 into Figure 2. Bucket B overflows and a split image, bucket B' , is created. There are two directory entries (001 and 101) that pointed to B before the split. Half of them (101) now points to the split image B' . The local depth of both buckets B and B' are increased by 1.

Discussion Deletion may cause a bucket to become empty, in which case it can be merged with its *buddy* bucket. The buddy bucket is referenced by the directory entry which shares the last (local depth-1) bits. For instance, the buckets referenced by direction entries 1111 and 0111 are buddy buckets. Many deletions can cause the directory to halve its size and thus decrease its global depth. This is triggered by the bucket merging which causes all local

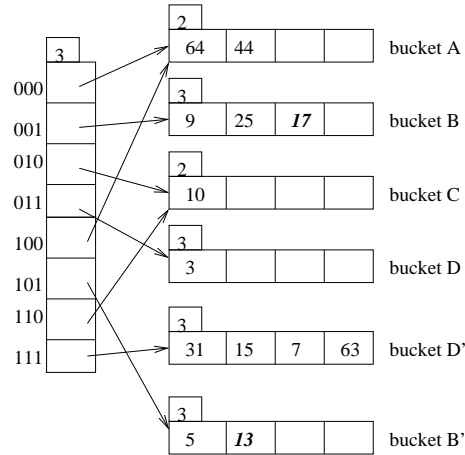


Figure 3: The directory does not double after inserting 17 and 13 into Figure 2.

depth to be strictly smaller than the global depth.

The fact that Extendible Hashing does not use any overflow page may significantly increase the directory size, as one insertion may cause the directory to double more than once. Consider the case when global depth is 3, and the bucket referenced by directory entry 001 overflows with five records 1, 17, 33, 49, 65. The directory is doubled. Directory entries 0001 and 1001 points to the overflowed bucket and the split image. All the five keys will remain in the original bucket which is again overflowing. Therefore the directory has to be doubled again. To alleviate this problem, one can allow a certain degree of overflow page links. For instance, whenever the fraction of buckets with overflow pages becomes larger than 1%, double the directory.

KEY APPLICATIONS

Extendible Hashing can be used in applications where exact match query is the most important query such as hash join [2].

CROSS REFERENCE

Hashing, Linear Hashing, Bloom Filter.

RECOMMENDED READING

- [1] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible Hashing: a Fast Access Method for Dynamic Files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.
- [2] D. A. Schneider and D. J. DeWitt. Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines. In *Proc. 16th International Conference on Very Large Databases (VLDB)*, pages 469–480, 1990.