Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS8391

DATA STRUCTURES (COMMON TO CSE & IT)

UNIT No. 5

5.6 REHASHING AND EXTENDIBLE HASHING

Version: 1.XX

REHASHING AND EXTENDIBLE HASHING

RE-HASHING SYSTEM:

Rehashing System increases the size of a hash table array, and restoring all of the items

into the array using the hash function.

When the original hash table is too full,

- Build the new hash table that is about twice as big (relatively next prime that is at least twice the current table's size) with an associated new hash function.
- Scan down the original hash table and compute the hash location for each element
- and Insert the elements into the new hash table. Then drop the original table.

Rehash Process:

The Rehashing process occurs when, the original hash table is HALF full an insertion fails load reaches a certain level (load factor) – best option for rehashing.

Load Factor – Number of Key elements in the hash Table and can be represented as λ (when λ = 0 (table empty); λ = 0.5 (half full); λ =1 (table Full)

Example: Hash the following key elements 18, 15, 6 and 24 for the TableSize of 7.

Hash (18) = 18 % 7 = 4

Hash (15) = 15 % 7 = 1

Hash (6) = 6 % 7 = 6

Hash (24) = 24 % 7 = 3

0	
1	15
2	
3	24
4	18
5	
6	6

Initiating the Rehashing

System, New hash table

Size = 7 * 2 = 14,

and the nearest PRIME greater than 14 is 17. By scanning down the original hash table and Rehashing using the new hash function as, Hash (18) = 18 % 17 = 1

Hash (15) = 15 % 17 = 15

Hash (6) = 6 % 17 = 6

Hash (24) = 24 % 17 = 7

Now the original Hash table is freed. New HashTable

0	
1	18
2	
3	
4	
5	
6	6
7	24
8	
9	
10	
11	
12	
13	
14	
15	15
16	

Pros and Cons of Rehashing System

 Rehashing can be used in other Data structures as well. For instance if the Queue data structure became full, declare a double-sized array and copy everything over, freeing the original.

 Rehashing frees the programmer from worrying about the table size.

Cons:

 Rehashing is time consuming, rehash every element

 once again. It is also very expensive when running short of memory space.

EXTENDIBLE HASHING SYSTEM :

Extendible hashing is a type of hash system which treats a hash as a bit string, and uses a try for bucket lookup. Because of the hierarchical nature of the system, re-hashing is an incremental operation (done one bucket at a time, as needed). This means that time-sensitive applications are less affected by table growth than by standard full-table rehashes. **Hash Function:** The hash function Hash (Key) for the extendible hash system returns a binary number. The first i bits of each string will be used as indices to figure out where they will go in the "directory" (hash table).

Additionally, i is the smallest number such that the first i bits of all keys are different.

Key terms used here is:

1. The key size that maps the directory (the Global depth), and
2. The key size that has previously mapped the bucket (the Local depth)

Operations on hash table:

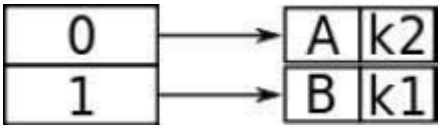
1. Doubling the directory when a bucket becomes full - If the local depth is equal to the global depth, then there is only one pointer to the bucket, and there is no other directory pointers that can map to the bucket, so the directory must be doubled.

2. Creating a new bucket, and re-distributing the entries between the old and the new bucket

- If the bucket is full, if the local depth is less than the global depth, then there exists more than one pointer from the directory to the bucket, and the bucket can be split

Example: Keys (say k1, k2, k3) to be used: 100100, 010110, 110110

Initially the bucket size is 1. The first two keys to be inserted, k1 and k2, can be



distinguished by the most significant bit, and would be inserted into the table.

Now, if k3 were to be hashed to the table, it wouldn't be enough to distinguish all three keys by one bit (because k3 and k1 have 1 as their leftmost bit. Also, because the bucket size is one, the table would overflow. Because comparing the first two most significant bits would give each key a unique location, the directory size is doubled to 4 as:

And so now k1 and k3 have a unique location, being distinguished by the first two leftmost bits. Because k2 is in the top half of the table, both 00 and 01 point to it because there is no other key to compare to that begins with a 0.