CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC
No.132, AECS Layout
IT Park Road, Bangalore 560 037
T: +91 80 28524466/77, F: +91 80 28524630
E: info@cmrit.ac.in | www.cmrit.ac.in

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

FS MODULE 5

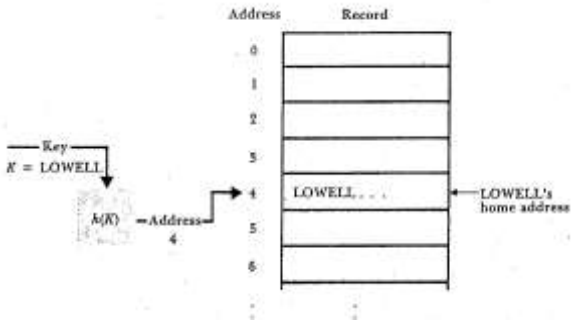
Prepared by: Prof. Prasad B S

1. **What is hashing? Explain a simple hashing algorithm.**
2. **What is collision? Explain collision resolution by progressive overflow.**
3. **Explain the different collision resolution techniques with an example.**
4. **Explain different methods used to avoid collision in hashing technique.**
5. **Suppose that 1000 addresses are allocated to hold 800 records in a randomly hashed file and that each address can hold one record. Compute the following values:**
   i) The packing density
   ii) The expected number of addresses with no records assigned to them.
   iii) The expected number of addresses with exactly one record assigned.
   iv) The expected number of addresses with one record or one or more synonyms
   v) The expected number of overflow records assuming that only one record can be assigned to each home addresses
   vi) Percentage of overflow records
6. **Explain the working of extendible hashing.**
7. **Write short notes on:**
   i) Dynamic hashing ii) Linear hashing
8. **Write short notes on Extendible hashing performance.**
9. **Write a note on buddy-buckets.**

Answers:

1. **What is hashing? Explain a simple hashing algorithm.**

Ans: HASH FUNCTION: A Hash function is like a black box that produces an address every time a key is dropped. The process of producing an address when a key is dropped to hash function is called Hashing. Formally hash function is given by h(K) that transforms a key 'K' into an address. The resulting address is used to store and retrieve the record.



A has function achieves the goal of file structure i.e. to access or retrieve any record in a single seek by converting the key in address and using direct access to fetch the record in single seek.

A simple hashing algorithm:

(Note: Though it can be any simple hashing function to convert a key to an address value, it is preferred to write the same algorithm which is given as example in the text book)

This algorithm has three steps:

1. Represent the key in numerical form.
2. Fold and add.
3. Divide by a prime number and use the reminder as the address.

Step 1: Represent the key in numerical form.

If the key is string of characters take ASCII code of each character and use it to form a number. Otherwise no need to do this step. Example if the key is LOWELL we will have to take ASCII code for each charaters to form a number. If the key is 550 then no need to do this step. We will consider LOWELL as the key.

LOWELL – 76 79 87 69 76 76 32 32 32 32 32 32

32 is ASCII code of blank space. We are padding the key with blank space to make it 12 characters long in size.

Step 2: Fold and add.

In this step Folding and adding means chopping off pieces of the number and add them together. Here we chop off pieces with two ASCII numbers each as shown below:

7679 | 8769 | 7676 | 3232 | 3232 | 3232

Integer values that exceed 32767 causes overflow errors or become negative. So addition of the above integer values results in 33820 causing an overflow error. Consequently we need to make sure that each successive sum is less than 32767. This is done by first identifying the largest single value we will ever add in summation and making sure after each addition the intermediate result differs from 32767 by that amount. Assuming that keys contain only blanks and upper case alphabets, so the largest addend is 9090 corresponding to ZZ. Suppose we choose 19937 as our largest allowable intermediate result. This differs from 32767 by much more than 9090, so no new addition will cause overflow. Ensuring of intermediate result not exceeding 19937 is done by using mod operator, which returns the reminder when one integer is divided by another.

7679 + 8769 = 16448 => 16448 mod 19937 => 16448

16448 + 7676 = 24124 => 24124 mod 19937 => 4187

4187 + 3232 = 7419 => 7419 mod 19937 => 7419

7419 + 3232 = 10651 => 10651 mod 19937 => 10651

10651 + 3232 = 13883 => 13883 mod 19937 => 13883

The number 13883 is the result of the fold and add operation.

Step 3: Divide by a prime number and use the reminder as the address.

In this step the number produced by the previous step is cut down so that it falls in the within the range of addresses of records in the file. This is done by dividing the number by address size of the file. The remainder will be home address of record. It is given as shown below

a = s mod n

If we decide to have 100 addresses i.e 0-99 than a= 13883 mod 100, which is equal to 83. So hash address of LOWELL will be 83.

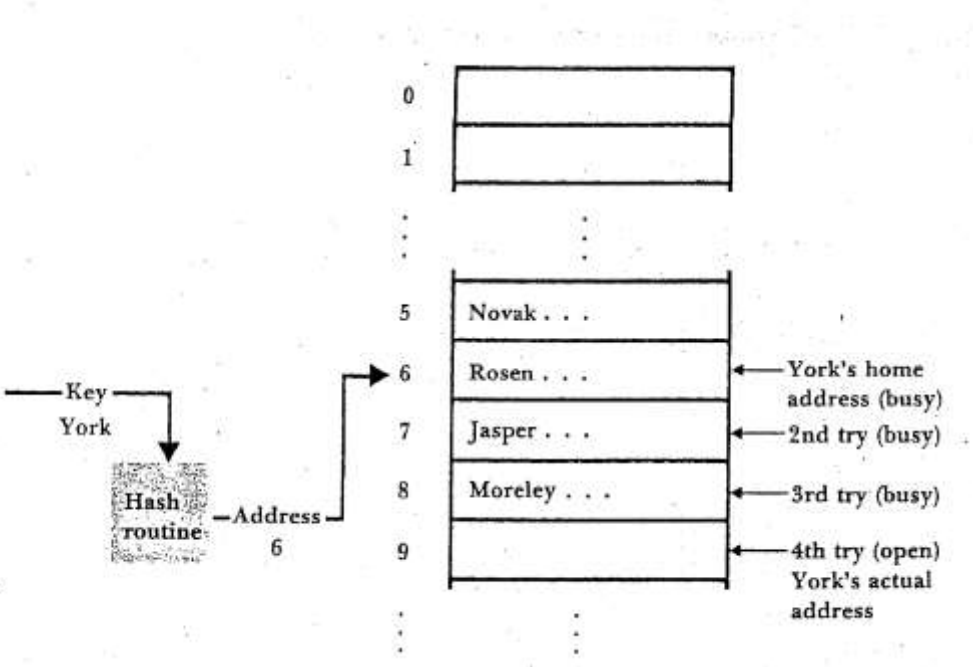
2. What is collision? Explain collision resolution by progressive overflow.

Ans:

Collision: An attempt to store a record at an address which does not have sufficient room ie already occupied by another record which is a synonym. A collision occurs when two record keys has to the same address.

Progressive overflow: If a key, k1, hashes into the same address, a1, as another key, k2, then look for the first available address, a2, following a1 and place k1 in a2. If the end of the address space is reached, then wrap around it. When searching for a key that is not in, if the address space is not full, then an empty address will be reached or the search will come back to where it began.

Progressive Overflow causes extra searches and thus extra disk accesses. If there are many collisions, then many records will be far from "home".



Search Length: Search length refers to the number of accesses required to retrieve a record from secondary memory. The average search length is the average number of times you can expect to have to access the disk to retrieve a record.

Average search length = (Total search length)/(Total number of records)

3. Explain the different collision resolution techniques with an example.

Ans: (Note: These techniques can be asked individually also)

Collision resolution techniques are:

a. Progressive overflow
b. Buckets
c. Double Hashing
d. Chained Progressive Overflow
e. Chaining with a Separate Overflow Area
f. Scatter Tables

a. Progressive overflow: refer previous question answer

b. Buckets: Buckets allows to store more than one record in a single hash address i.e. storing More than One Record per address. A bucket describes a block of records sharing the same address that is retrieved in one disk access. When a record is to be stored or retrieved, its home bucket address is determined by hashing.

Effect of bucket on collision:

To compute how densely packed a file is, we need to consider
1) the number of addresses, N, (buckets)
2) the number of records we can put at each address, b, (bucket size) and
3) the number of records, r.
Then,
Packing Density = r/bN

Though the packing density does not change when halving the number of addresses and doubling the size of the buckets, the expected number of overflows decreases dramatically.

Disadvantage: When a bucket is filled, we still have to worry about the record overflow problem, but this occurs much less often than when each address can hold only one record.

c. Double Hashing:

Double hashing is similar to progressive overflow. The second hash value is used as a stepping distance for the probing. The second hash value should never be one. The second hash value should be relatively prime to the size of the table. \

If there is collision at the hash address produced by h1(k) = Say 'X', then the key is dropped to second hash function to produce address 'C'. The new record is stored at the address 'X+C'. A collision resolution scheme which applies a second hash function to keys which collide, to determine a probing distance 'C'.

h1(k)= X (collision at X)
h2(k)= C
X+C = home address of k

The use of double hashing will reduce the average number of probes required to find a record.

d. Chained Progressive Overflow: Chained progressive overflow forms a linked list, or chain, of synonyms. Each home address contains a number indicating the location of the next

record with the same home address. The next record in turn contains a pointer to the other record with the same home address. Assume the following has address generation:

Key	Home address
Adams	20
Bates	21
Cole	20
Deans	21
Evans	24
Flint	20

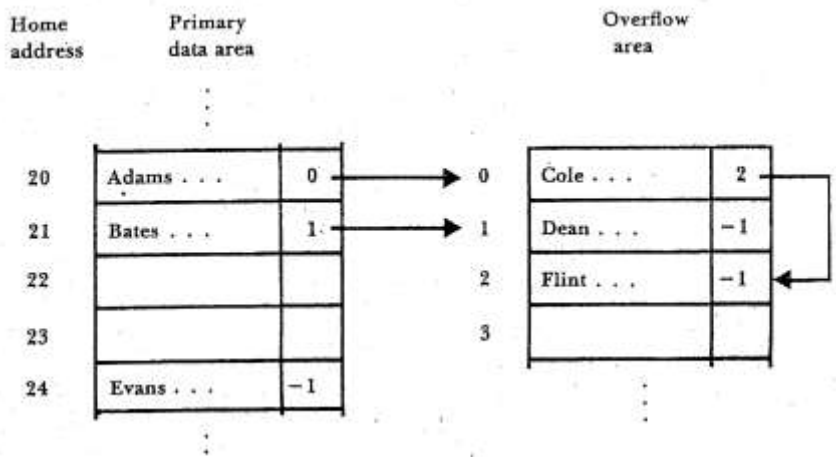
The records with same home address are linked as shown in the figure below: In the figure below Adams contain the pointer to Cole which is synonym. Then Bates contain pointer to Dean which are again synonym.

Address	Key	Next record
20	Adams	22
21	Bates	23
22	Cole	25
23	Deans	-1
24	Evans	-1
25	Flint	-1
26		
.		
.		

e. Chaining with a separate overflow area:

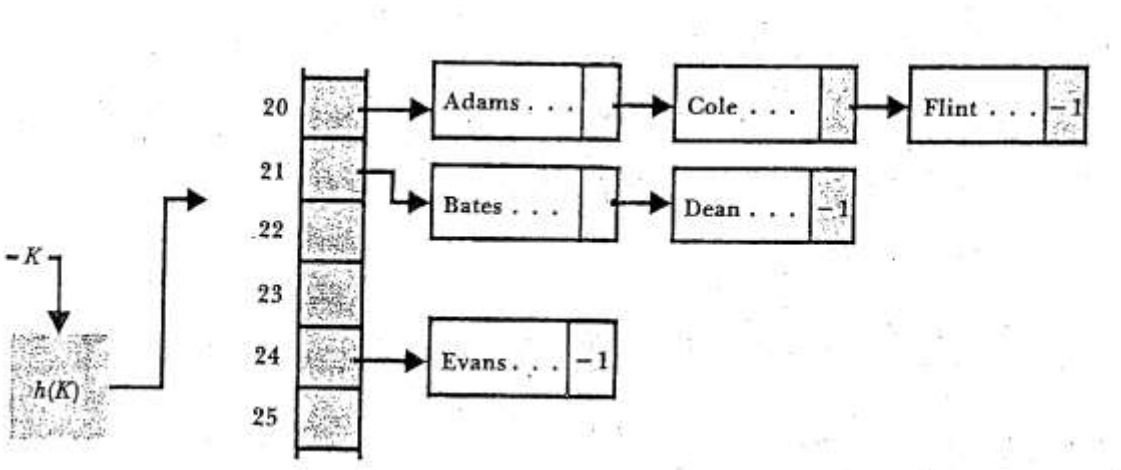
One way to keep overflow records from occupying home addresses where they should not be is to move them all to a separate overflow area. The set of home addresses is called prime data area, and the set of overflow addresses is called the overflow area.

Whenever a new record is added if its home address is empty, it is stored in primary storage area. Otherwise it is moved to overflow area, where it is added to a linked list that starts at home address.



f. Scatter tables: If all records are moved into a separate "overflow" area, with only links being left in the hash table, the result is a scatter table. In scatter table records are not stored at the hash address. Only pointer to the record is stored at hash address.

Scatter tables are similar to index but they are searched by hashing rather than some other method which is used in case of indexing. Main advantage of scatter tables is they support use of variable length records.



4. Explain different methods used to avoid collision in hashing technique.

Ans: (Note: Do not get confused this is how to avoid collision from happening rather than resolving collision after it occurs)

It is near to impossible to find a perfect hashing algorithm which do not cause any collision. So a more practical solution is to reduce the number of collision to an acceptable level. Some of the ways to reduce the collusion to acceptable level are:

1. Spread out the records.

2. Use extra memory.

3. Put more than one record at a single address.

Spread out the records: Collision occur when two or more records compete for the same address. If we could find hashing algorithm that distributes the records fairly randomly among the available addresses, then we would not have large number of records clustering around certain address.

Use extra memory: It is easier to find a hash algorithm that avoids collisions if we have only a few records to distribute among many addresses than if we have about the same number of records as addresses. Example chances of collisions are less when 10 records needs to be distributed among 100 address space. Whereas chances of collision is more when 10 records needs to be distributed among 10 address space. The obvious disadvantage is wastage of space.

Put more than one record at a single address: Create a file in such a way that it can store several records at every address. Example, if each record is 80 bytes long and we create a file with 512 byte physical records, we can store up to six records at each address. This is called as bucket technique.

5. **Suppose that 1000 addresses are allocated to hold 800 records in a randomly hashed file and that each address can hold one record. Compute the following values:**
   i) The packing density
   ii) The expected number of addresses with no records assigned to them.
   iii) The expected number of addresses with exactly one record assigned.
   iv) The expected number of addresses with one record or one or more synonyms
   v) The expected number of overflow records assuming that only one record can be assigned to each home addresses
   vi) Percentage of overflow records

Ans:

i) Packing density:
Given by : No of records/ No of address
800/1000=0.8

ii) Probability of X number of records hashed to a given address is given by poisson function

$$p(x) = \frac{(r/N)^x \, e^{-(r/N)}}{x!}$$

Where r/N is the packing density.

P(0)= (0.8)0 X e-(0.8)/0!= .449*1000=449

iii) P(1)= (0.8)1 X e-(0.8)/1!=.313*1000=313

iv) P(2)= (0.8)2 X e-(0.8)/2!=.095*1000=95
P(3)= (0.8)3 X e-(0.8)/3!=.038*1000=38
P(4)= (0.8)4 X e-(0.8)/4!=0.007*1000=7
P(5)= (0.8)5 X e-(0.8)/5!=0.001*1000=1
address with one or more synonyms=95+38+7+1=141

v) (95*1)+(38*2)+(3*7)+(1*4)=196

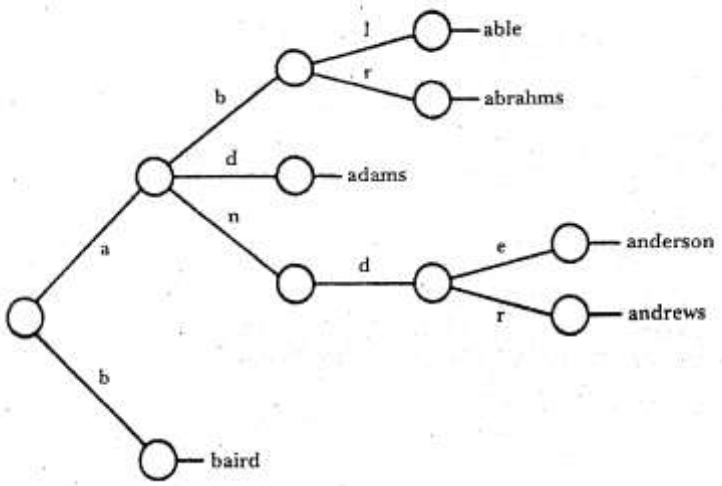
vi) 196/800*100= 24.5%

6. **Explain the working of extendible hashing**

Ans:

Extendible Hashing: The key idea behind extendible hashing is to combine conventional hashing with another retrieval approach called the **trie**. Tries are also sometimes referred to as radix searching because the branching factor of the search tree is equal to the number of alternative symbols (the radix of the alphabet) that can occur in each position of the key.

Suppose we want to build a trie that stores the keys able, abrahms, adams, anderson, andrews, and baird. A schematic form of the trie is shown below.

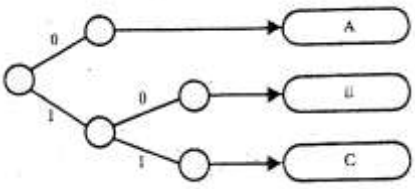


The searching proceeds letter by letter through the key. Because there are twenty-six symbols in the alphabet, the potential branching factor at every node of the search is twenty-Six.In searching a trie we sometimes use only a portion of the key. We use more of the key as we need more information to complete the search. This use-more-as-we-need-more capability is fundamental to the structure of extendible hashing.

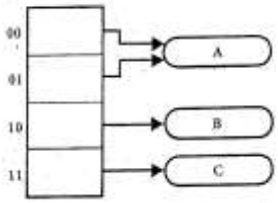
Representing a trie:

Rather than representing the trie as a tree, we flatten it into an array of contiguous records, forming a directory of hash addresses and pointers to the corresponding buckets.

Step 1: The first Step in turning a tree into an array involves extending it so it is a complete binary tree with all of its leaves at the Same level as shown in figure below. Even though the initial 01s enough to select bucket A, the new form of the tree also uses the second address bit so both alternatives lead to the same bucket.

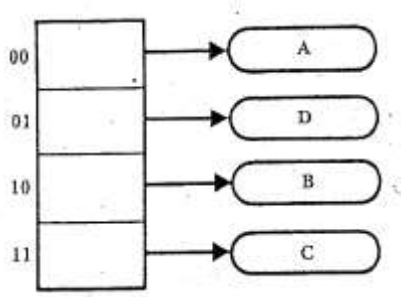


Step 2: Once we have extended the tree this way, we can collapse it into the directory structure shown in Fig below. Now we have a Structure that provides the kind of direct access associated with hashing: given an address beginning with the bits 10, the 1O2th directory entry gives us a pointer to the associated bucket.

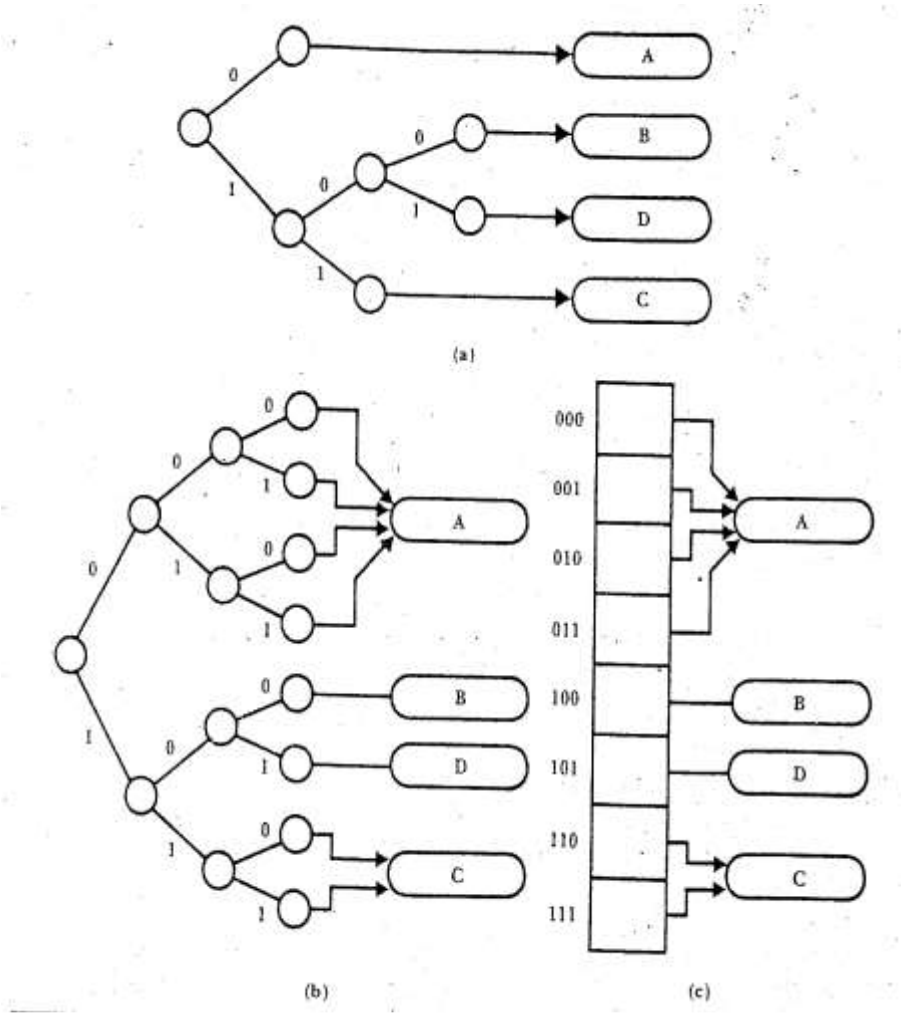


Handling overflow:

Case 1: Overflow in bucket A causing split, can use unused bit space address to address bucket D



Case 2: Furter split in bucket B, no unused bit space available requiring to use 3 bit hash address as shown below, extending address space dynamically:



7. Write short notes on:

i) Dynamic hashing ii) Linear hashing

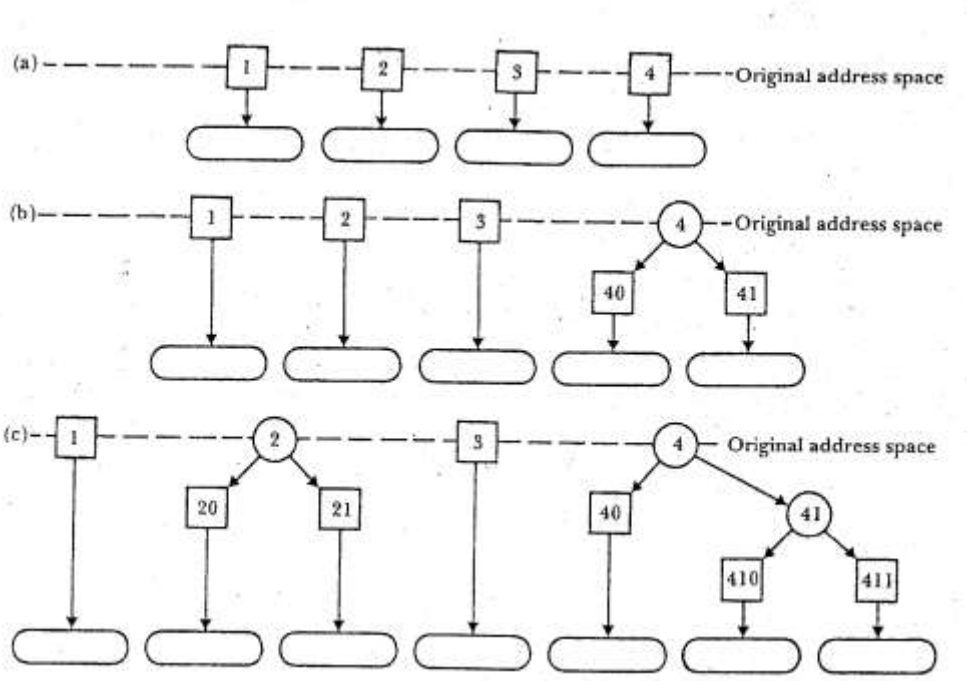
Ans:

Dynamic Hashing: Functionally, dynamic hashing and extendible hashing are very similar. Both use a directory to track the addresses of the buckets, and both-extend the directory through the use of tries.

The key difference between the approaches is that dynamic hashing, like conventional, static hashing, starts with a hash function that covers an address space of a fixed Size. As buckets within that fixed address space overflow, they split, forming the leaves of a trie that grows down from the original address node. Eventually, after enough additions and splitting, the buckets are addressed through a forest of tries that have been seeded out of the original static address space.

Let's look at an example. Figure (a) shows an initial address space of four and four buckets descending from the four addresses in the directory. In Fig. (b) we have split the bucket at address 4. We address the two buckets resulting from the split as 40 and 41 . We change the shape of the directory node at address 4 from a square to a circle because it has changed from an external node. In Fig (c) we split the bucket addressed by node 2, creating the new external nodes 20 and 21. We also split the bucket addressed by 41 , extending the trie downward to include 410 and 411. Finding a key in a dynamic hashing scheme can involve the use of two hash functions rather than just one. First, there is the hash function that covers the original address space. If you find that the directory node is an external node and therefore points to a bucket, the search is complete.

However, if the directory node is an internal node, then you need additional address information to guide you through the ls and 0s that form the trie. The primary difference between the two approaches is that dynamic hashing allows for slower, more gradual growth of the directory, whereas extendible hashing extends the directory by doubling it.



Linear Hashing:

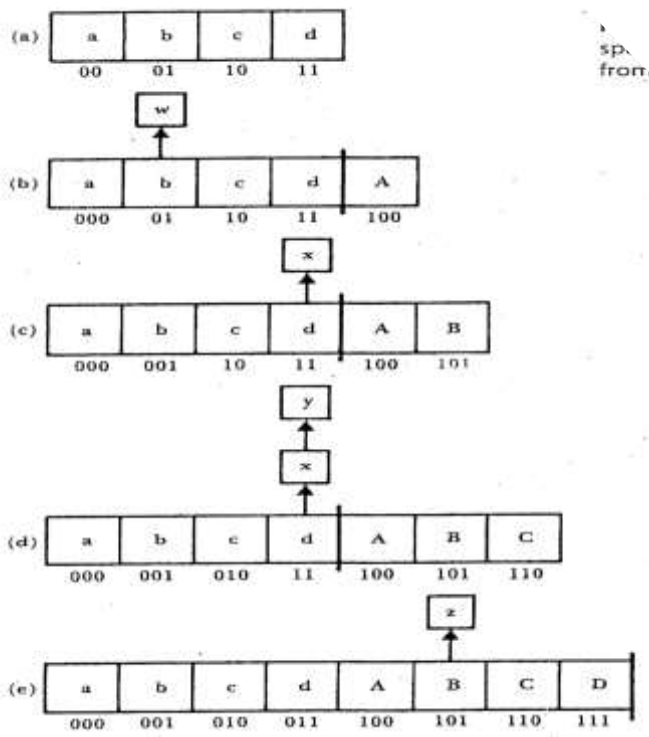
Linear hashing, does away with the directory. Linear hashing, like extendible hashing, uses more bits of hashed value as the address space grows. Note that the address space consists of four buckets rather than four directory nodes that can point to buckets.

As we add records, bucket b overflows. The overflow forces a split. However, as Fig. (b) Shows, it is not bucket b that splits, but bucket a. The reason for this is that we are extending the address space linearly, and bucket a is the next bucket that must split to create the next linear extension, which we call bucket A. A 3-bit hash function, h3(k), is applied to buckets a'and A to divide the records between them. Since bucket b was not the bucket that we split, the overflowing record is placed into an overflow bucket w.

We add more records, and bucket d overflows. Bucket b is the next one to split and extend the address space, so we use the h3(k) address function to divide the records from bucket b and its overflow bucket between b and the new bucket B. The record overflowing bucket d is placed In an overflow bucket x. The resulting arrangement is illustrated in Fig. (c).

Figure (d) shows what happens when, as we add more records, bucket d overflows beyond the capacity of the overflow bucket w. Bucket c is the next in the extension sequence, so we use the h3(k)address function to divide the records between c and C.

Finally, assume that bucket B overflows. The overflow record is placed in the overflow bucket z. The overflow also triggers the extension to bucket D, dividing the contents of d, x, and y between buckets d and D. At this point all of the buckets use the h3(k) address function, and we have finished the expansion cycle. The pointer for the next bucket to be split returns, to bucket a to get ready for a new cycle that will use an h4(k) address function to reach new buckets.



8. Write short notes on Extendible hashing performance.

Ans:

Performance is measured in terms of time and space.

Time efficiency: If the directory for extendible hashing can be kept in memory, a single access is required to retrieve the record. If the directory is so large that it must be paged in and out of memory, 2 access may be necessary. Access performance for extendible hashing is O(1) if the directory can be kept in memory. If the directory must be paged off to the disk, worst case performance is O(2).

Space efficiency: Two use of spaces

1. For buckets
2. For the directory

For Buckets: Space utilization is strongly periodic and fluctuates between 0.53 and 0.94. For a given number of records r and block size b , the average number of blocks N approximated by the formula

$$N \approx \frac{r}{b \ln 2} N$$

Space utilization or packing density is given by

$$\text{Utilization} = \frac{r}{bN}$$

Substituting for N gives Utilization = 0.69 , that is average space utilization is 69 percent.

For Directory: Estimated directory size is

$$\text{Estimated directory size} = \frac{3.92}{b} r^{(1 + 1/b)}$$

9. Write a note on buddy-buckets.

Ans:

If extendible hashing is to be a truly dynamic system like B-trees or AVL trees, it must be able to shrink files gracefully as well as grow them. If deletions cause a bucket to be substantially less than full, we can find a buddy bucket to collapse with.

Two buckets are buddies if:

◆ They are at the same depth.

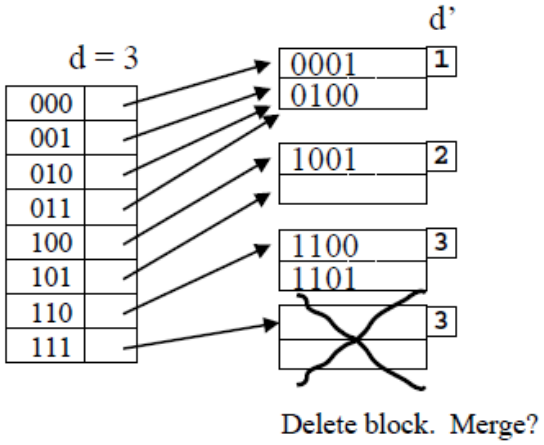
◆ Their initial bit strings are the same.

◆ Collapsing them will fit all records in one bucket.

We can definitely collapse if a bucket is empty. When all buckets have d' < d then we can halve the address space.

Example:

Delete 1111.



After Merge blocks and shrink directory :