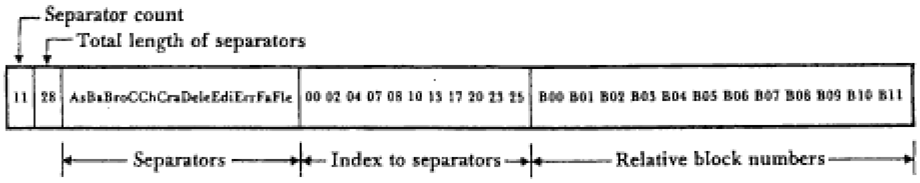
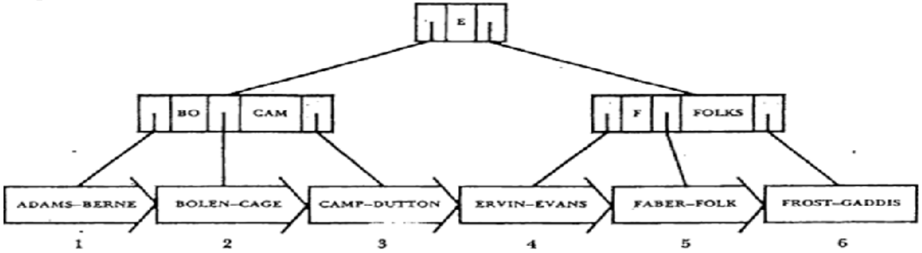
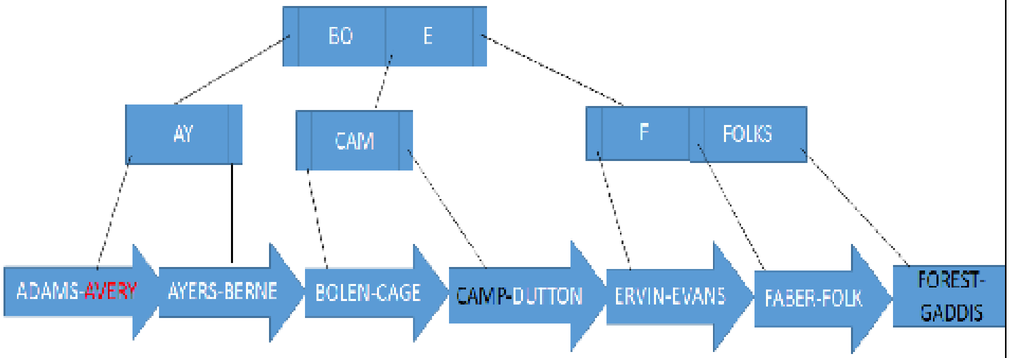
USN

CELEBRATING 25 YEARS
CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

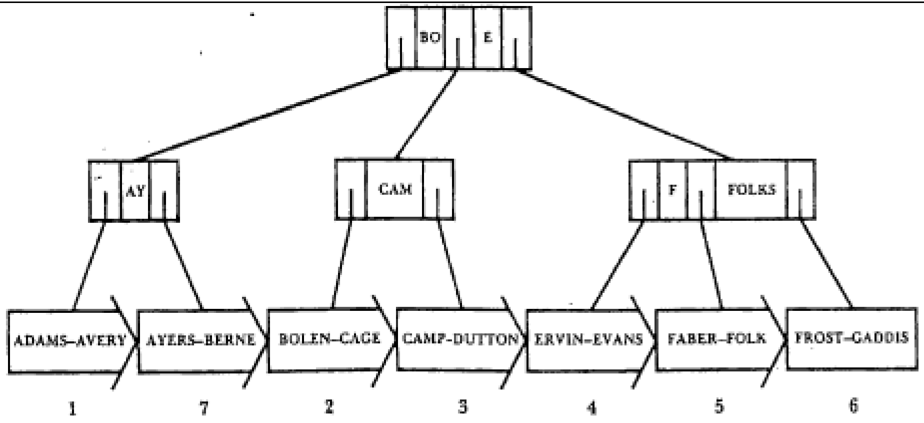
Internal Assessment Test 3 – July 2022
QP Set 3

Sub:	File Structures					Sub Code:	18IS61	Branch:	ISE
Date:	08/07/2022	Duration:	90 min’s	Max Marks:	50	Sem/Sec:	VI A, B & C		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	**Distinguish B-Trees, B+ Trees, and Simple Prefix B+ Trees with suitable example and sketch.** B-Tree is a data structure used to access or retrieve data very quickly from file or database. Majorly B-tree is used in indexing which can lead the path to where the actual records stored. To access quickly the desired record, and avoid delay by means of imbalance in height, and store multiple keys in each node, B-tree is best to implement for indexing. B-tree are tree arranges the records in B-Tree order based the key. The data we can able to read from the root node to leaf nodes where actually record found in B-tree. Where as in B+ tree, the actual records stored (or available) at leaf nodes and all the leaf nodes are connected. The root node and intermediate nodes contains only the key which is used to retrieve actual data stored in the bottom level leaf nodes. Prefix B+ tree on the other hand, instead of key, the part of the key or prefix of the key will be taken to establish index. Using the prefix of the key, we can construct the index to locate the data stored in the sequence set which arranged at the bottom level leaf.	10	CO3	L1
2.	**With neat diagram explain internal structure of Index set block to maintain variable order prefix B+ tree.** • The Important Issue is, size and structure of the index set nodes • In our example, Small index set nodes and have treated them as fixed-order B-tree nodes, but the separators are variable in length. • The physical size of a node for the index set is, same as the physical size of a block in the sequence set. • So, we can say, Index set block • A common block size make it easier to implement a buffering to create a virtual simple prefix B+tree, • Finally, the index set block and sequence set blocks are often mingled within the same file to avoid seeking between two separate files. AaBaBroCChCraDeleEdiErrFaFle 00 02 04 07 08 10 13 17 20 23 25 Concatenated separators Index to separators **Separator Count:** we need this to help us find the middle element in the index to the separator so we can begin our binary search **Total length of separator :** the list of merged separator varies in length from block to block. Since the index to the separators begins at the end of this variable-length list, we need to know how long the list is so we can find the beginning of our index.	10	CO3	L2

	 Separator count Total length of separators 11 28 AsBaBroCChCraDeleEdiErrFaFle 00 02 04 07 08 10 13 17 20 23 25 B00 B01 B02 B03 B04 B05 B06 B07 B08 B09 B10 B11 Separators Index to separators Relative block numbers			
3.	**Discuss in details with neat diagram about maintenance of prefix B+ tree block overflow while deleting or inserting.** Changes are Localized to Single Blocks in the Sequence Set Delete the Records - EMBRY and FOLKS * Deleting these records, not required merging and redistribution with in the sequence set  E BO CAM F FOLKS ADAMS–BERNE BOLEN–CAGE CAMP–DUTTON ERVIN–EVANS FABER–FOLK FROST–GADDIS 1 2 3 4 5 6 **Changes Involving Multiple Blocks in the Sequence Set** What happens, If the insertion or deletion of records change the number of blocks in the sequence set? Consider the block is full, again we are adding new records, and the record to be stored in the block which is already full. Similarly, we deleted a record from the block, which makes the block with less entries  BO E AY CAM F FOLKS ADAMS-AVERY AYERS-BERNE BOLEN-CAGE CAMP-DUTTON ERVIN-EVANS FABER-FOLK FOREST-GADDIS **Changes Involving Multiple Blocks in the Sequence Set** Addition and Deletion of records make changes on multiple blocks in the Sequence Set. * Eg., Insertion into first block causes the block to split. * The new block is linked into the correct position in the sequence set, such that following block 1 and preceding block 2.	10	CO3	L2

<table>
<tr><td></td><td>



</td><td></td><td></td><td></td></tr>
<tr><td>4.a</td><td>

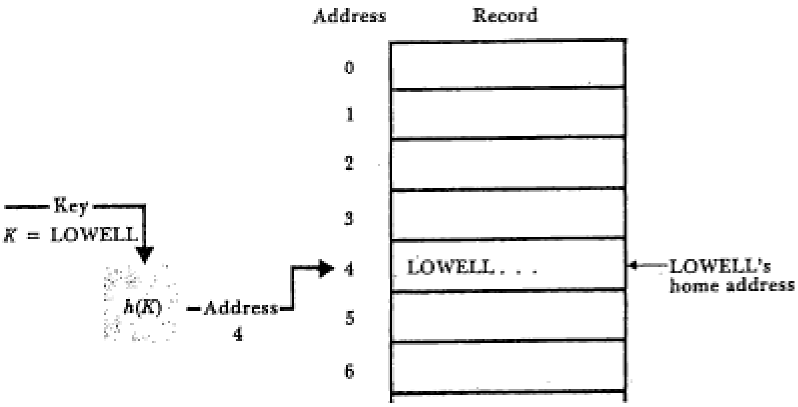
What is hashing? Brief about different hashing methods.

- A hash function is like a black box that produces an address every time you drop in a key.
- More formally, it is a function h(k) that transforms a key K into an address.
- With hashing, the address generated appear to be random,- there is no immediately obvious connection between the key and the location of the corresponding record, even though the key is used to determine the location of the record. For this reason, hashing is sometimes referred to as randomizing.
- With hashing, two different keys may be transformed to the same address, so two records may be sent to the same place in the file.
- Hashing uses a repeatable pseudorandom function. The hashing function should produce a uniform distribution of hash values.

home address

The address produced by the hashing of a record key. Under ideal conditions, hashing provides O(1) key driven file access.

Collisions: If any two key produce the same address then, it called collision. Example, the key OLIVIER and LOWELL, produce the same address, because both the key having common first two letters.



</td><td>5</td><td>CO3</td><td>L1</td></tr>
<tr><td>4.b</td><td>

What is collision? Explain collision resolution by progressive overflow.

Collisions: If any two key produce the same address then, it called collision. Example, the key OLIVIER and LOWELL, produce the same address, because both the key having common first two letters.

Any hashing program must incorporate some method for dealing with records that cannot fit into their home addresses.

Progressive Overflow Works

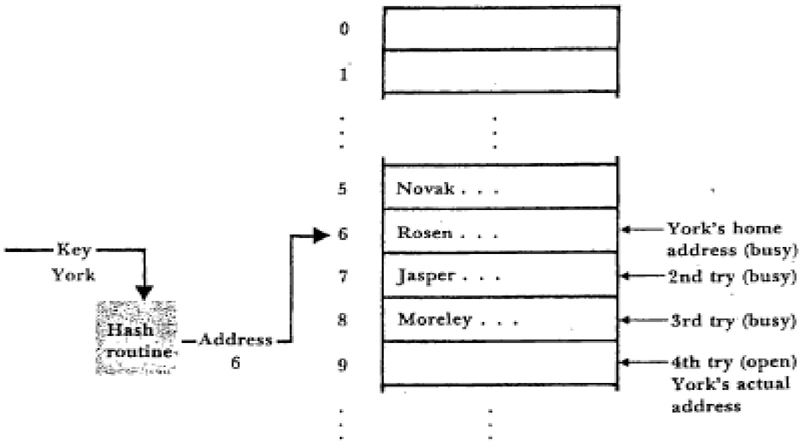
</td><td>5</td><td>CO3</td><td>L2</td></tr>
</table>

Consider, we want to store the record ‘York’, but its hash address as the same for ‘Rosen’.

If progressive overflow is used, the next several addresses are searched in sequence until an empty one is found. The first free address becomes the address of the record.

If we need to find York’s record in the file. Since York still hashes to 6, the search for the record begins at address 6. it does not find York’s record there, address 9 is the first record found empty, so the record pertaining to York is stored in address 9.
If an open address is encountered, the searching routine might assume this means that the record is not in the file; or

If the file is full, the search comes back to where it began. Only then is it clear that the record is not in the file. When this occurs, or even when we approach filling our file, searching can become intolerably slow, whether or not the record being sought is in the file

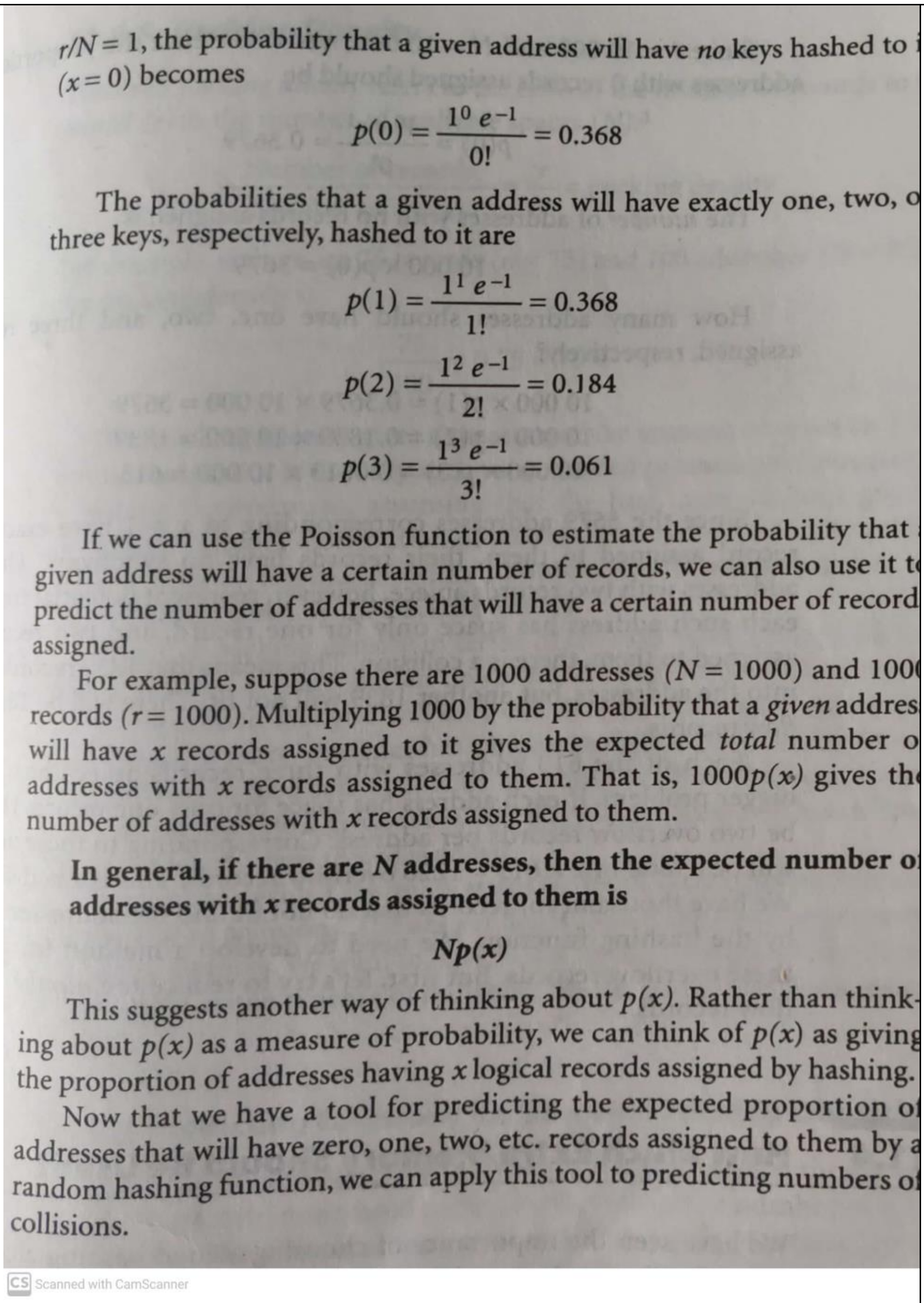


Search Length

The reason to avoid overflow is, of course, that extra searches have to occur when a record is not found in its home address. If there are lot of collisions, there are going to be a lot of overflow records taking up spaces where they ought not to be.

The term search length refers to the number of accesses required to retrieve a record from secondary memory

5.	**Illustrate the probability distribution of Records using Poisson distribution method for the following** • None will hash to the given address? • Exactly one key will hash to the address? • Exactly two keys will hash to the address • Exactly three, four, and so on keys will hash to the address? • All keys in the file will has to the same given address?	10	CO3	L3



$r/N = 1$, the probability that a given address will have *no* keys hashed to i ($x = 0$) becomes

$$p(0) = \frac{1^0\, e^{-1}}{0!} = 0.368$$

The probabilities that a given address will have exactly one, two, o three keys, respectively, hashed to it are

$$p(1) = \frac{1^1\, e^{-1}}{1!} = 0.368$$

$$p(2) = \frac{1^2\, e^{-1}}{2!} = 0.184$$

$$p(3) = \frac{1^3\, e^{-1}}{3!} = 0.061$$

If we can use the Poisson function to estimate the probability that given address will have a certain number of records, we can also use it t predict the number of addresses that will have a certain number of record assigned.

For example, suppose there are 1000 addresses ($N = 1000$) and 100 records ($r = 1000$). Multiplying 1000 by the probability that a *given* addres will have x records assigned to it gives the expected *total* number o addresses with x records assigned to them. That is, $1000p(x)$ gives th number of addresses with x records assigned to them.

In general, if there are N addresses, then the expected number o addresses with x records assigned to them is

$$Np(x)$$

This suggests another way of thinking about $p(x)$. Rather than thinking about $p(x)$ as a measure of probability, we can think of $p(x)$ as giving the proportion of addresses having x logical records assigned by hashing.

Now that we have a tool for predicting the expected proportion o addresses that will have zero, one, two, etc. records assigned to them by a random hashing function, we can apply this tool to predicting numbers o collisions.

6. **What is packing density? How much extra memory should be used to avoid collision? Illustrate the following points with your examples.**

- How many addresses should have one record plus one or more synonyms?
- Assuming that only one record can be assigned to each home address, how any overflow records can be expected?
- What percentage of records should be overflow records?

10 CO3 L3

Packing Density

The term packing density refers to the ratio of the number of records to be stored (r) to the number of available spaces(N)^3

$$\frac{\text{Number of records`}}{\text{Number of spaces}} = \frac{r}{N} = \text{packing density}$$

- How many addresses should have no records assigned to them?

Np(0)=1000x(0.5)0 e^0.5/0!

= 1000x0.607
=607

- How many addresses should have exactly one record assigned?
  Np(1)=1000x(0.5)^1 e^-0.5

``````````````

1!

- ***How many addresses should have one record plus one or more synonyms?***

The values of p(2), p(3), p(4), and so on give the proportion of addresses with one, two, three, and so on synonyms assigned to them

p(2)+p(3)+p(4)+p(5) = 0.0758+0.0126…
= 0.0902

***Assuming that only one record can be assigned to each home address, how many overflow records can be expected?***

For each address represented by p(3), one record can be stored at the address, two are overflow records, and so on. Hence, the expected number of overflow records is given by

1xNxp(2)+2xNx(3)…..=Nx[1+p(2) + 2xp(3)+3x Nxp(4)….=107

***What percentage of records should be overflow records?*** If there are 107 overflow records and 500 records in all, then the proportion of overflow record is

***107/500 = 0.124=21.4%***

Faculty Signature CCI Signature HOD Signature
``````````````