Solutions (one out of numerous possible) – Hashing

1. Consider the following partial table of an ordered library catalogue:

| **Author_ID** | **Book_ID** | **Author_Name** | **Book** |
|---|---|---|---|
| LBa_001 | Ba001_How | Lisa Feldman Barrett | How Emotions are Made |
| ADa_001 | Da001_Sel | Antonio Damasio | Self Comes to Mind |
| EBl_012 | Bl012_Mal | Enid Blyton | Malory Towers |
| MMi_009 | Mi009_Emo | Marvin Minsky | Emotion Machine |
| VRa_001 | Ra001_Pha | Vilayanur Ramachandran | Phantoms in the Brain |
| JRo_015 | Ro015_Fan | Joanne Rowling | Fantastic Beasts and Where to Find Them |
| JRo_015 | Ro015_Qui | Joanne Rowling | Quidditch Through the Ages |
| PWo_015 | Wo015_Pot | Pelham Grenville Wodehouse | The Pothunters |
| PWo_015 | Wo015_Wod | Pelham Grenville Wodehouse | Wodehouse at the Wicket |

*Note:*

- It is known that most user queries involve the Author_Name and/or Book_Name.
- Typing the first three characters <xxx> of a search value for either of the attributes, on the library exploration portal, is sufficient to retrieve a set of records that begin with <xxx>.
- Clearly state all assumptions.

*Questions:*

a) Use extendible hashing with bucket size 4 to design an effective access strategy for the above?

Understandably, the database would have one index structure on the Author_Name, another index structure on the Book_Name, and a third on a combination of the Author_Name AND Book_Name to facilitate retrievals as described.

Assuming a scenario where the user enters the first three characters of the Author_Name AND the Book_Name, we demonstrate below a possible extendible hashing structure to deal with the situation.

We consider here the use of a composite index_key comprising <Author_Name, Book_Name> of the form <xxxyyy | xxx represents the first three characters of the Author_Name and yyy represents the first three characters of the Book_Name>. Ideally the keys would be tuned to both the upper- and lower-cases of the alphabets; for simplicity, we shall be working with lower-case alphabets.

Thus, the keys for the database entries would be as follows:

Assumption: Prepositions, conjunctions and articles not considered as part of Index_Key

| Index_Key | ASCII conversion of Index_Key excluding 3 MSB | Author_Name | Book_Name |
|---|---|---|---|
| barhow | 00010 00001 10010 01000 01111 10111 | Barrett | How Emotions are Made |
| damsel | 00100 00001 01101 10011 00101 01100 | Damasio | Self comes to Mind |
| blymal | 00010 01100 11001 01101 00001 01100 | Blyton | Malory Towers |
| minemo | 01101 01001 01110 00101 01101 01111 | Minsky | Emotion Machine |
| rampha | 10010 00001 01101 10000 01000 00001 | Ramachandran | Phantoms in the Brain |
| rowfan | 10010 01111 10111 00110 00001 01110 | Rowling | Fantastic Beasts and Where to Find Them |
| rowqui | 10010 01111 10111 10001 10101 01001 | Rowling | Quidditch Through the Ages |
| wodpot | 10111 01111 00100 10000 01111 10100 | Wodehouse | The Pothunters |
| wodwod | 10111 01111 00100 10111 01111 00100 | Wodehouse | Wodehouse at the Wicket |

If *d* = global depth, we have assumed here a hash function *h*, such that *h*(index) = *d* LSBs each of the first alphabet of the first (for author name) and fourth characters (for book name) forming the index. Therefore, the extendible hashing strategy with bucket size = 4 on the presented data would lead to the following index-storage arrangement:

| Global Directory (depth = 1 for each Index_Key section) | Local Directory (depth = 1; bucket size = 4) |
|---|---|
| 0 0 | <barhow>; <rampha>; <rowfan> |
| 0 1 | <damsel>; <blymal>; <rowqui> |
| 1 0 | <wodpot> |
| 1 1 | <minemo>; <wodwod> |

(Note: Typically, *h* would be updated to include other bits within a character-representation as well as other characters in the index over time; one may assume any hash function of their choice)

b) Use linear hashing with bucket size 4 to design an effective access strategy for the table provided?

Here, Bucket_size ($m$) = 4. Therefore, using the same index_keys as those in the previous answer, and using the hash_function ($h_0$) = *index_key* % *m*:, the linear hash-table storage allocation is as follows: (*index_key* = position of the first alphabet in the index_key)

| Index_Key | First alphabet position |
|---|---|
| barhow | 2 |
| damsel | 4 |
| blymal | 2 |
| minemo | 13 |
| rampha | 18 |
| rowfan | 18 |
| rowqui | 18 |
| wodpot | 23 |
| wodwod | 23 |

Stage_1 of insertion (bucket split pointer ($p$) = 0):

| Bucket_No | Data Directory | Overflow Directory |
|---|---|---|
| 0 | <damsel>; | |
| 1 | <minemo>; | |
| 2 | <barhow>; <blymal>; <rampha>; <rowfan> | |
| 3 | | |

Stage_2 of insertion (Bucket_No. 0 is split into Bucket_No. 0 & Bucket_no. 4; $p$ = 1; $h_1$ = *index_key* % 8 for entries in Bucket_No. 0 & Bucket_no. 4; other buckets follow $h_0$>

| Bucket_No | Data Directory | Overflow Directory |
|---|---|---|
| 0 | <damsel>; | |
| 1 | <minemo>; | |
| 2 | <barhow>; <blymal>; <rampha>; <rowfan> | <rowqui> |
| 3 | <wodpot>; <wodwod> | |
| 4 | | |

c) For the record <MMi_009, Mi009_Soc, Marvin Minsky, Society of Mind>: Trace the insertion into the structures of Q.a and Q.b. Compare between the two placements.

For the new record <MMi_009, Mi009_Soc, Marvin Minsky, Society of Mind>, the index entries are of the form:

| Index_Key | ASCII conversion of Index_Key excluding 3 MSB | Author_Name | Book_Name |
|---|---|---|---|

| | | | |
|---|---|---|---|
| minsoc | 01101 01001 01110 10011 01111 00011 | Minsky | Society of Mind |

Leading to the final allocation in the extendible hash-bucket as follows:

| Global Directory (depth = 1 for each Index_Key section) | Local Directory (depth = 1; bucket size = 4) |
|---|---|
| 0 0 | <barhow>; <rampha>; <rowfan> |
| 0 1 | <damsel>; <blymal>; <rowqui> |
| 1 0 | <wodpot> |
| 1 1 | <minemo>; <wodwod>; **<minsoc>** |

No further bucket splits are observed; however the entries within a local directory are not arranged in the alphabetical order.

The allocation in the linear hash-bucket for <minsoc> is as follows:

| Bucket_No | Data Directory | Overflow Directory |
|---|---|---|
| 0 | <damsel>; | |
| 1 | <minemo>; **<minsoc>** | |
| 2 | <barhow>; <blymal>; <rampha>; <rowfan> | <rowqui> |
| 3 | <wodpot>; <wodwod> | |
| 4 | | |

Some observations:
1. At this stage, the linear hash-table has suffered a bucket split – though with no entries in the new bucket space
2. The extendible-hash technique is semantically aligned to the purpose of the particular index_key design
3. Data directories for neither of the techniques are lexicographically sorted => the in-directory search time for a record/index_key is of the order of a linear search operation
4. Operational complexity for the extendible-hashing technique includes conversion of index_keys into their ascii equivalent and selection of the bits of consequence at every stage of data allocation. The linear-hashing technique, on the other hand, incurs additional space requirements in the form of overflow pages.
5. The extendible-hashing technique would have to involve further operations if duplicate index_keys (for different <author, book> tuples) were to form