ASSIGNMENT - II

Page No.
Date :

1) What is meant by an index? Explain the operations required to maintain the index file

Index

- An index is a table containing a list a key and corresponding reference fields. The reference field (address) points to the record where the information referenced by the key is found.

(OR)

- An index is a tool for finding records in a file it consists of
  - → key field on which the index is searched
  - → Reference field that tells particular address of the key.

Operations required to maintain the index file

- Create the original empty index and data file
- Load index file into memory before using it
- Remote the index file from memory after using it
- Add data records to data file
- Delete records in data file
- Update records in data files

→ **Creating a file**

- Two files must be created, a data file to hold the data objects and index file to hold primary key index, both are initially empty

→ **Loading the index into memory**

- Only the index file is loaded into memory

- The loading into memory can be done sequentially reading a large number of index records
- Using the index file, the required record is search & the corresponding record in the file is read through the reference in index file

→ Rewritting the index into memory
- The index file that was loaded on to memory needs to be rewritten back to the index file on the close operation
- the changed index file has to be rewritten back to the memory. This action is important to ~~guard~~ guard against the power failure, the operator turning the machine off at the wrong time & other such disaster

→ Record addition
- Adding a new record to data file requires that we also add an entry to the index
- In data file, record can be added anywhere, however the byte offset of new record should be saved
- Since the index is kept in sorted order by key insertion of new index entry probably requires some rearrangement of index

→ Record deletion
- To delete a record we must just remove the corresponding index entry from index and the space created can be filled up by the shifting the below entries to close up the space
- Since the record deletion takes place in memory, record shifting is not too costly

Page No.
Date :

→ Record updating

- Record updating falls in two categories

i) → The update changes the value of the field
  - Both index and data file may need to be updated, and index file has to be sorted

ii) → The update does not affect the key field
  - Doesn't require rearrangement of index file
  - If record size is unchanged or decreased by the update, the record can be written directly into old space
  - If record size is increased by updat, a new slot for record will have be found

Q 14) What are inverted lists? How does it improve the secondary index structure?

- Inverted lists are indexes in which a key is associated with a list of reference fields.
- Secondary index structure results in 2 distinct difficulties
  → Addition of new record with secondary key - We have to rearrange the index file everytime a new record is added to the file even if new record is for an existing secondary key.
  → Redundant secondary key - If there are duplicate secondary keys, the secondary key field is repeated for each entry and space is wasted larger index files are less likely to fit in memory
- Solution for these difficulties is created of files such as secondary indexes, in which a secondary key leads to a set of one or more primary keys, called inverted lists

Page No.
Date :

- There are two solutions

i) Solution 1

-Change the secondary index structure so it associates an array of references with each secondary key.

Eg Chethan 1VA09IS015 1VA09IS019

-fig. shows secondary key index containing space for multiple reference for each secondary key.

Revised student Index

Secondary key	Set of primary key references
Amar	1VA09IS001
Bharath	1VA09IS010
Chethan	1VA09IS015 1VA19IS018 1VA09IS019
Gayatri	1VA07IS023
Mary	1VA09IS045

ADV.

Avoids the need to rearrange the secondary index file until a new secondary key is added

DISADV

May restrict the number of references that can be associated with each secondary key.

Cause internal fragmentation.

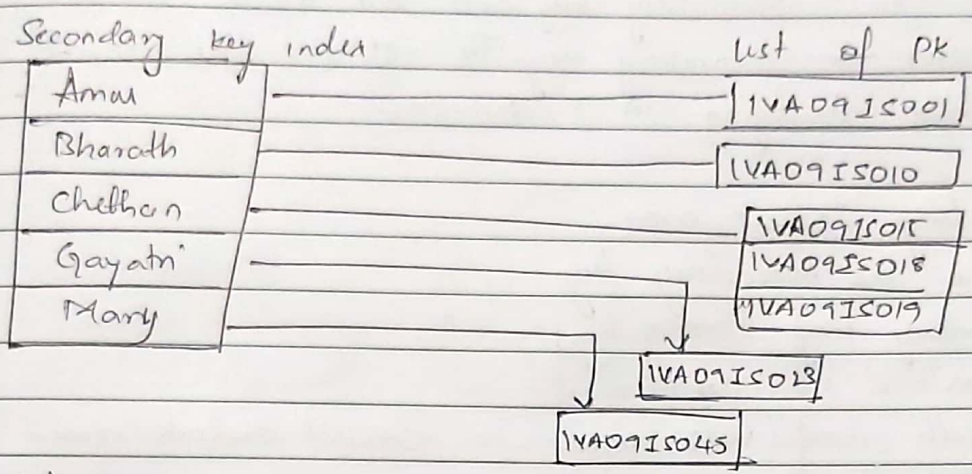
ii) Solution 2

Linking the list of reference (to other solution)

Method: Each secondary key point to a different list of PK references. Each of these lists could grow to be as long as it needs to be and no space would be

Page No.
Date :

Lost to internal fragmentation



Advantages

- No wastage of space due to internal fragmentation
- When a new record is added, the new PK in secondary index file are stored & arranged

Dicadv

- A large number of small files are required to store the list of PK.

3) Illustrate with an example Application of Secondary Indexing (Retrival using combination of secondary keys). Explain the solution by using "Linking the list of reference" technique

Retrival Using Combinations of Secondary keys

- One important application of secondary keys involves using two or more of them in combination to retrieve special subset of records from the data file
- With secondary keys, we can search for things like
  - → Records with name 'chethan'
  - → Records with semester 'I'
  - → Records with name 'Chethan' and semester 'I'

Page No.
Date :

Suppose, there are two secondary indexes, one containing name as a secondary key and the other containing semester as secondary key. To retrieve the record of Chethan studying in I semester.

Secondary Indexes are-

Name as Secondary key.

Secondary key	Primary key
Amar	1VA09IS001
Bharath	1VA09IS010
Chethan	1VA09IS015
Chethan	1VA09IS016
Chethan	1VA09IS019
Gayatri	1VA09IS022
Manoj	1VA09IS045

Selected students named 'Chethan'

Chethan	1VA09IS015
Chethan	1VA09IS016
Chethan	1VA09IS019

Semester as Secondary key

Secondary key	Primary key
I	1VA09IS001
I	1VA09IS010
I	1VA09IS015
II	1VA09IS018
II	1VA09IS017
III	1VA09IS023
IV	1VA09IS045

Selected students of I semster

I	1VA09IS001
I	1VA09IS010
I	1VA09IS015

The student name & semester required coincides with the record having PK as 1VA09IS015. So from the primary index the byteoffset is retrieved for that pk to retrieve the record

Page No.
Date :

Linking the list of reference [Refer Q2 solution 2]

16) 4) Explain Record addition, deletion and updating operation with respect to indexing by multiple keys.

i) Record addition

- When a secondary index is used, adding a record involves updating the data file, the primary index, and secondary index. Secondary index update is similar to primary update
- Similar to primary indexes, cost of doing this greatly decreases if the secondary indexes can be read into memory and changed there
- Secondary keys are stored in canonical form
- fig shows sec. key index organised by name

Title Index

Secondary key	Primary key
Amar	1DT18IS001
Bharath	1DT18IS010
Chethan	1DT18IS015
Chethan	1DT18IS018
Chethan	1DT18IS019
Gayatri	1DT18IS023
Mary	1DT18IS045

- In the eg above, there are 3 records with the name 'Chethan'. Within the group, they should be ordered according to the values of the reference field (PK)

Page No.
Date :

ii) Record Deletion

- Removing a record from data file means removing the corresponding entry in primary index and all the entries in secondary indexes that indexes that refer to this primary index entry.
- Like primary index, the secondary indexes are maintained in sorted order by key. Deleting an entry would involve rearranging the remaining entries to close up the space left open by deletion
- Thus deleting a record consumes more time for it secondary index file.
- This can be avoided by deleting corresponding entry of only the primary index file. The secondary index file is not changed. Thus eliminating the modification and rearrangement in secondary index
- When a record is searched using secondary key, the key is searched in secondary index & corresponding primary key is found. The corresponding entry is not found in primary index file, and the search ends, informing the user that the record does not exists.

iii) Record updating

There are 3 possible situations

a) Update changes the secondary key:

— We may have to rearrange the secondary key so it stays in sorted order.

b) Update changes the PK:

Has large impact (or changes) on PK index but often requires that we update only the affected PK

in all secondary index also. If the secondary key of the affected primary key is the same, then sorting is done according to the primary key.

(iii) c) Update confirmed to other fields:

No changes necessary to secondary indexes. But the primary index file will change if the address of modified record changes

20) Discuss the problem of statement ~~of~~ to keep index on secondary storage? Explain indexing with binary search tree.

- If the index too large, then: Index access and maintenance must be done ~~of~~ on secondary storage

Disadvantage/Problem of storing index file in secondary memory:

- Binary Searching of index requires several seeks rather than being performed at memory speed
- Index rearrangement (due to record addition/deletion) requires shifting or sorting records on secondary storage, this is extremely time consuming

[OR]

- Searching the index using binary search requires multiple access to secondary memory.
- Insertion and deletion must be done to secondary memory and sorting of data is to be done

Indexing with Binary Search Tree

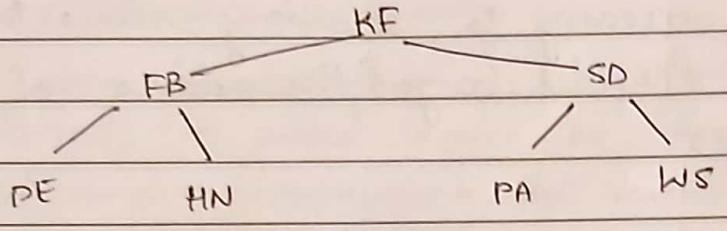
To overcome the 2nd problem, the key can be represented in Binary Search Tree.

If the keys are in sorted list, the middle element is taken as the

root and a balanced tree is formed.

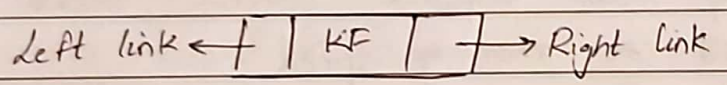
Eg - If the list is-
DE, FB, HN, KF, PA, SD, WS

If KF is ~~the~~ taken as root, a balanced tree is obtained

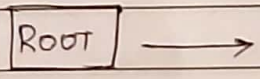


Using tree structure - there is no problem of sorting the file (when new records are added). The new key is simply linked to the appropriate leaf node

- In a Binary Search Tree, each node will have three information :- key-values, left link, right link



The above tree is represented in memory as follows:



	KEY	LEFT CHILD	RIGHT CHILD
0	FB	2	4
1	PA		
2	DE		
3	SD	1	6
4	HN		
5	KF	0	3
6	WF		

Page No.
Date :

Suppose a key tf is added to the three, the key will be attached to the rightmost subtree i.e. As the right child of "WF"

Record contents ofor a linked representation of binary tree will be:-

ROOT ⟶ 5

	KEY	LEFT CHILD	RIGHT CHILD
0	FB	4	4
1	PA		
2	DE		
3	SD	1	6
4	HN		
5	KF	0	3
6	WF		7
7	XF		

As a key are added to Binary Search Tree, it may become imbalanced. A tree is balanced when the difference in height of the shortest path to the tree and the longest path of the tree is not more than one level

or

A tree is balanced when all the leaves of the tree are at the same level or differ by one level

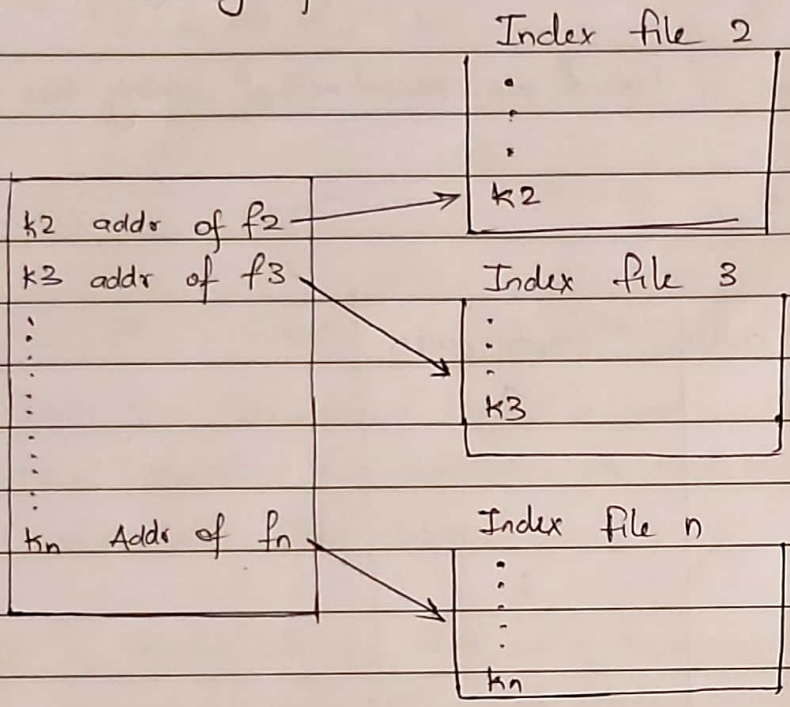
2) What is multilevel indexing? Explain the concept of B-tree in multilevel indexing with an example.

Multilevel Indexing

- Multilevel index is index of the index file
- Since index records form a sorted list of keys, one of

Page No.
Date :

keys in each index record (usually largest key) is taken as the key of whole record



Multilevel indexes help to reduce the number of disk accesses and their overhead space costs are minimal. An index file to another index file is multilevel indexing.

B-Trees

B-trees are multilevel ~~inheritance~~ indexes that solve the problem of linear cost of insertion and deletion. Each node of a B-tree is an index record. Each of these nodes has a same maximum number of key-reference pairs, called the order of B-tree

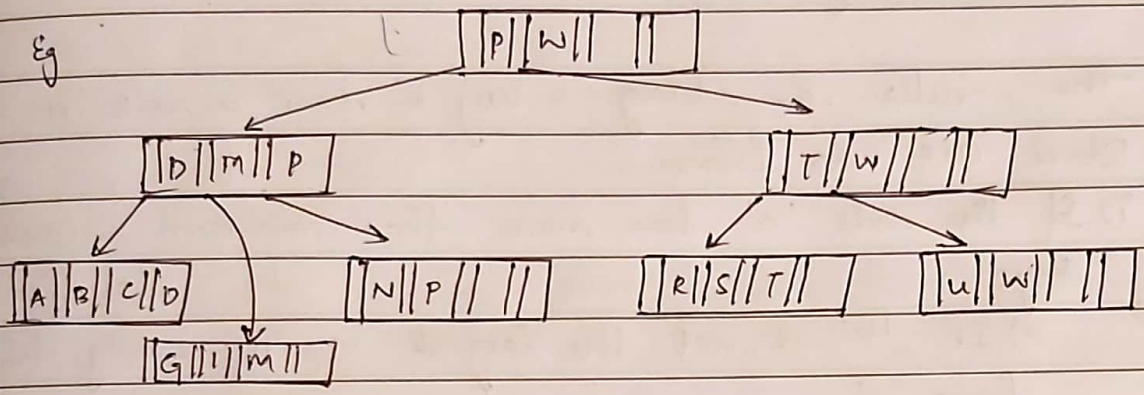
Order of a B-tree is the maximum no. of keys in a node of a tree.

(or)

Maximum no of descendants from a node. It is usually denoted by 'm'

While inserting a new key to a node-
- If the node is not full, simply insert the key
- If the node is full, the node is split into two nodes, each with half of the keys.

Eg

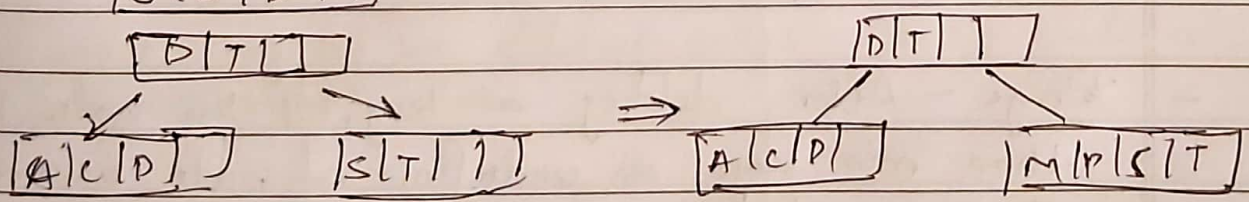


Creation of B Tree

If the order of B-tree is 'm', each node in a B-tree can store maximum of m keys.
- Initially the keys are added to the node
- When the node is full, it is split into two nodes, each containing half of keys
- When new node is created, the largest key of new nodes must be inserted into the higher level node key promotion
- Continue the process of insertion, splitting ~~the~~ and key promotion until the elements to be inserted into B tree is complete

Eg C S D T A M P

C S D T → no problem in insertion



Page No.
Date :

22) Explain deletion, merging.. and redistribution of elements in B-tree

- The deletion of a key from a node, affects the sibling node. Node that have same parent node are called sibling nodes

- The rules for deleting a key 'k' from a node 'n' in a Btree are as follow -

i) If node 'n' has more than minimum number of keys -

a) If 'k' is not the largest in 'n' - simply delete 'k' from 'n'

b) If 'k' is the largest in 'n' delete 'k' and modify the higher level index to reflect the new largest key in 'n'

ii) If 'n' has exactly min. number of keys and one of the sibling of 'n' has -

a) few enough keys (min no of keys only), merge 'n' with its sibling and delete a key from the parent node

b) Extra key, redistribute by moving some keys from a sibling to 'n', and modify the higher level indexes to reflect the new largest keys in the affected nodes

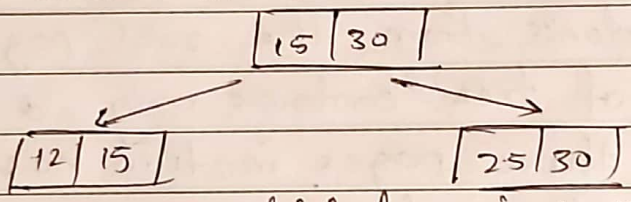
- Merge - After deleting a keys, if the node has less than min keys an underflow of node occurs. So the keys of this node can be put into a sibling

node. This process is called merging

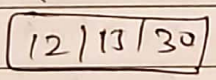
Redistribution - After deleting a key from a node, if underflow occurs in a node and if there is no space for merging the keys among the sibilings, i.e. if the siblings are full
The keys are distributed among the siblings, this process is called redistribution

Redistribution differs from both splitting (creating a node) and merging (deletion a node) there the number of nodes in a not changed

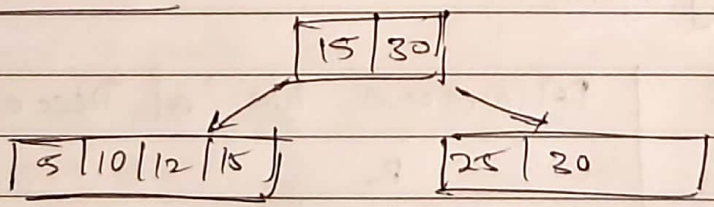
Example of deletion in B-Tree



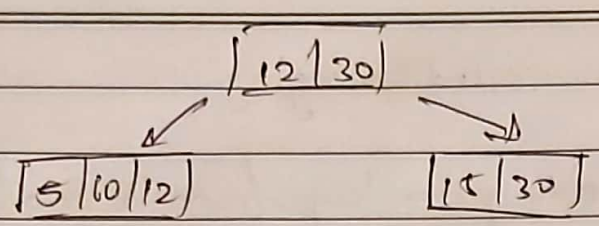
Suppose key to be deleted is '25', then the node is merged to form a single



for redistribution



Suppose key to be deleted is '25'. As 1st node is full, redistribution takes place.



23a) Explain worst case search depth of B tree

- The maximum number of disk accesses required to locate a key in the tree is nothing but worst case search. Every key appears in the leaf level
- Finding the depth of tree is nothing but worst case search. The worst case occurs when every page of tree has only one minimum no of descendants.
- In such case the keys are spread over a maximal height for the tree and a minimal breadth
- For a B-tree of order 'm' the minimum number of descendants from the root page is two, so 2nd level of tree contains only 2 pages.
- Each of these pages in turn has at least ceiling function of m/2 given by $\lceil m/2 \rceil$ descendants.
- The general pattern of the relation between depth and the minimum number of descendants takes the following form:

Level	Minimum No. of Descendants
1 (root)	2
2	$2 * \lceil m/2 \rceil$
3	$2 * (\lceil m/2 \rceil)^2$
4	$2 * (\lceil m/2 \rceil)^3$
d	$2 * (\lceil m/2 \rceil)^{d-1}$

Page No.
Date :

So in general for any level d of B-tree, the min. no of discendants extending from that level is

$$2 * (\lceil m/2 \rceil)^{d-1}$$

For a tree with 'N' keys in its leaves, we express the relationship between keys and the minimum height d as

$$\geq 2 * \lceil m/2 \rceil^{d-1}$$

Solving for d we arrive at the expression

$$d \leq 1 + \log_{\lceil m/2 \rceil} (N/2)$$

23b) Illustrate with an example the working of paged binary tree

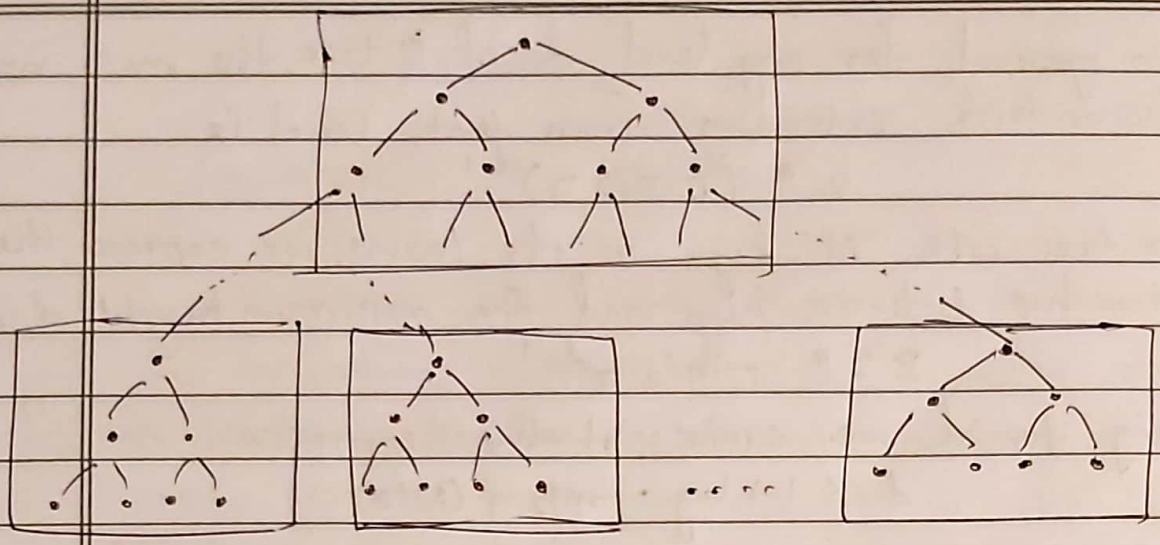
Paged Binary Tree

The main problem with binary search tree is that the disk utilization is extremly inefficient. Each disk read has only one node of data consisting of only three useful pieces of information - the key, address of left and right subtree

Each read operation reads one page of data i.e. 512 bytes. One page consists of one node of info, the remaining space is wasted.

This can be solved by storing more than one node in a single page. So that in one seek that many roles of info. can be retreived

For eg - if 7 nodes are stored in page, by accessing this page, we get the info of 7 nodes in a single access & single access will return the info. required for next move also, thus reducing cost of disk access.



Thus in one access we get the info. of 7 nodes & in 2 access we get the info of 63 nodes

Worst case comparsion blw binary tree and paged binary tree is shown below:

Binary tree: $\log_2(N+1)$ N - No of keys

Paged binary tree: $\log_{K+1}(N+1)$ K - No of keys on a page

If the no of keys (N) is 13421

Binary search requires $\log_2(13421+1) = 14$ seeks

Paged binary tree requires $\log_{7+1}(13421+1) = 5$ seeks

Advantages

- Seek time required to access many nodes is reduced compared in binary search tree

Disadvantage

- Most of the info. read is unused. All the data in page is read and thus there is wastage of transfer time

24) Write short note on B-tree Nomenclature? Discuss B tree properties?

Page No.
Date :

Nomenclature

Order of B-tree

According to Bayer, Mc Creight and Come r order of the B-tree is minimum number of keys that can be in a page of the tree

Knuth definition of order of the B-tree is maximum number of descendants that a page can have

Leaf

Bayer and Mc Creight refer to the lowest level of keys in a B-tree as the leaf level.

Knuth considers the leaf of a B-tree to be one level below the lowest level of keys. In other words, he considers the leaves to be actual data records that might be pointed to by lowest level of the keys in the tree

B-tree properties

For a B-tree of order m,

- Every page has a maximum of m descents
- Every page, except for the root and the leaves, has a maximum of $\lceil m/2 \rceil$ descendants
- The root has a maximum of 2 descendants (unless it is a leaf)
- All of the leaves are on the same level
- The leaf level forms a complete, ordered index of the associated data file