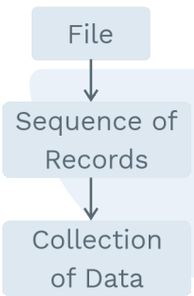
5 File Organization & Indexing

5.1 FILE ORGANIZATION AND INDEXING

Introduction:

- In a system, a database is stored in the form of files containing records.
- These files of records are stored in secondary memory like magnetic disks.
- Each file is partitioned into fixed-length blocks.
- One block can contain more than one data item.
- One of the reasons to use a database system is to decrease the number of blocks that are transferred between disk and memory.

File organization:



- A file is organized as a sequence of blocks.
- A block is organized as a sequence of records.
- Records means a collection of data items(fact).

Primary file organization:

- It describes how can we place file records on the disk as well as how can we access them based on the primary field.

Secondary file organization:

- In this, we can access the records of files based on alternate fields (not primary fields).

Note:

File organization refers to the organization of the data of a file into records, blocks and access structure.

- As we know, a block is the smallest unit that is used to transfer the data between disk and memory.

- So, we need to allocate records of the file to disk blocks.
- When block size > record size, then one block can have many records.
- But, we can not fit these records to one block if the number of records present are large.

Grey Matter Alert!

- The file is termed as fixed-length records where all the records are of exact same length.

Blocking factor:

It is defined as the maximum number of records that can be stored in a disk block.

Let us consider, BS > RS where,

BS represents block size in bytes and

RS represents size of the fixed length records.

Average number of records per block is known as Blocking factor.

We can define blocking factor = $\lfloor BS/RS \rfloor$ records per block. $\lfloor\ \rfloor$ represents floor function.

Note:

Sometimes record size does not divide block size exactly.

So, there will be some unused space = (BS- (Blocking factor * RS)) bytes.

- Records can be organized in a database file using the following two strategies:
  1) Spanned
  2) Unspanned

1) **Spanned:** In this strategy, records are allowed to span in more than one block. i.e. it allows partial part of records to be stored in a block.
Example: Consider the figure given below, which is depicts the spanned strategy for storing files of records into a block.

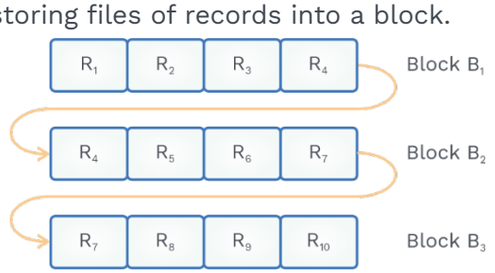


Fig. 5.1

In this example, record R_4 is stored in block B_1 as well as Block B_2. Similarly record R_7 is stored in block B_2 as well as B_3.

Advantage of spanned strategy:
No wastage of memory.

Disadvantage of spanned strategy:
Number of block accesses to access a record increases.

Note:

This strategy is appropriate for storing those records whose length is variable.

2) **Unspanned strategy:**
The records are not allowed to cross block boundaries, i.e. if no record can be stored in more than one block then this strategy is known as unspanned strategy.
Example: Consider the following example given below that depicts unspanned strategy for storing records.

Fig. 5.2

Advantage of unspanned strategy:
Number of block accesses to access a record reduces.

Disadvantage of unspanned strategy:
Wastage of memory.

Note:

This strategy is appropriate for storing those records whose length is fixed.

Note:

If the average record size is large, it is beneficial to use spanned organization, to reduce the space that is lost in each block.

- The Number of blocks b required for a file of a 'r' records

$$b = \lceil (r/\text{blocking factor}) \rceil \text{ blocks}.$$

Where ⌈ ⌉ is a ceiling function and it rounds the value up to the next integer.

SOLVED EXAMPLES

Q1 **Consider a file of size 2 MB, having 2K fixed length Blocks. Following are the records which need to be stored in the blocks: R1(500B), R2(800B), R3(300B) R4(1000B), R5(500B). If spanned organization of records is used. What are the total number of Blocks required to store all the records?**
1) 5 blocks 2) 4 blocks 3) 6 blocks 4) 3 blocks

Sol: **2)**

In spanned organization records are allowed to span in more than one Block.

$$\text{Block size} \frac{\text{File size}}{\text{Number of Blocks in File}}$$

$$= \frac{2 \times 2^{20}\text{B}}{2 \times 2^{10}} = 1\text{KB}$$

500+800+300+1000+500 = 3100 bytes
So, number of blocks required = ⌈3100 bytes/1024 bytes⌉
= ⌈ 3.02 ⌉
= 4
Hence, 4 Blocks are required to store all the records.

Allocation of file blocks on disk:

- File blocks can be allocated on the disk using the following techniques:
- **Continuous allocation:** In continuous allocation, the file blocks are allocated to consecutive disk blocks.
- **Linked allocation:** In linked allocation; one file block is linked to the other(next) file block with a pointer.
- **Disadvantage:** This allocation makes it difficult and slow to read the whole file.
- **Indexed allocation:** In Indexed allocation, one or more index blocks contain pointers that point to the existing file-blocks.

Files of unordered records:

- All records in file are inserted wherever there is place available for the record.
- There is no ordering of records.
- Files of unordered records are also known as Heap files.
- Any particular record is searched using linear search.
- **Advantage:** Inserting a record is efficient.
- **Disadvantage:** Searching for a record is inefficient; it is expensive procedure due to the involvement of linear search.

Note:

- For a file having 'b' blocks, on average, this requires searching (b/2) blocks.

Files of ordered records:

- **Advantage:**
  1) As there is no sorting required, we can efficiently read the records.
  2) There is no requirement for additional block access.
  3) Binary search is used to search for a record in a file that consists of records in an ordered way, so access of to records will be faster.

Note:

A binary search access $\log_2(b)$ blocks whether the record is found or not.

Average block access required for a file having 'b' blocks in basic file organization

	Types of Organization	Access/Search Method	Average Blocks To Access A Specific Record
1)	Heap (unordered)	Linear Search (Sequential scan)	b/2
2)	Ordered	Linear Search (Sequential scan)	b/2
3)	Ordered	Binary Search	$\log_2 b$

Table 5.1

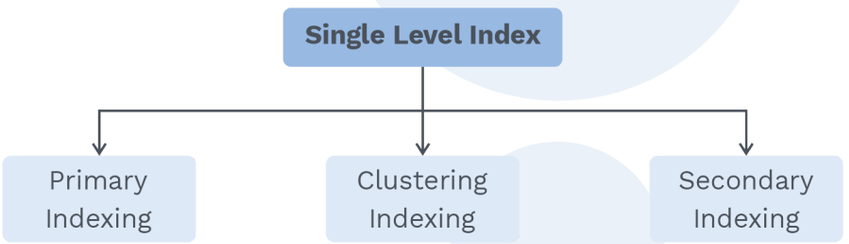
Index structure:

- Indexes are the auxiliary access structure.
- It is used to increase search efficiency.
- Using the index structure, we can easily access the records without creating any disturbance to the records that are placed physically in the main data file on disk.
- We can access records efficiently based on the indexing fields.
- Following are the different types of indexes used
  1) Single-level index
  2) Tree data structures
  (multilevel index, B+ trees).

Grey Matter Alert!

- There is another type of primary organization which is based on hashing.
- It provides very fast access to records under certain conditions.
- This organization is called the hash file.

5.2 SINGLE LEVEL INDEX



- An index structure is based on a field termed as indexing field.
  1) Primary index:
- It is an ordered-file where length of the records are fixed and contains 2 fields.
- The 1st field consists the data type, which is same as the primary key of the file where data is present.
- The second field consists a pointer pointing to a disk block.
- For every block present in the data file, there must be an index record in index file.

Definitions

"A primary index is specified on the ordered key field of an ordered file of records."

Note:

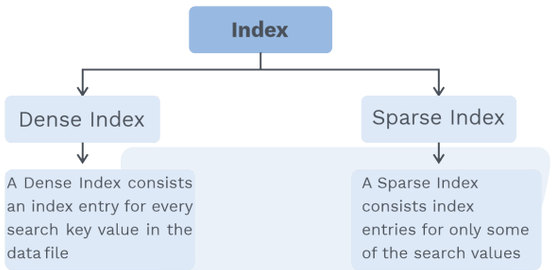
A Number of index-entries =Number of disk blocks.

- The average number of block accesses using primary index = $\lceil \log_2 Bi \rceil + 1$. where Bi refers to the total block count.

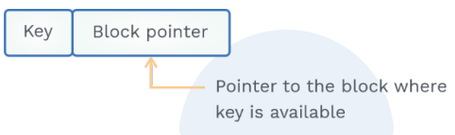
Note:

- Hey learners!!
  Do you know what an anchor record is?
  An anchor record is the first record in each block of the data file.
- Anchor record is also called as block anchor.

- Indexes are also characterized into two categories dense and sparse index.



- Structure of index files:
  Index record consists of two fields: key and block pointer.



- A sparse index stores lesser count of entries in comparison to the count of records in the original database file.

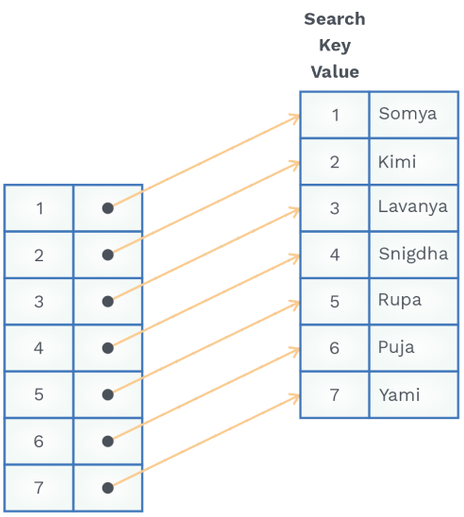


Fig. 5.3 Dense Index

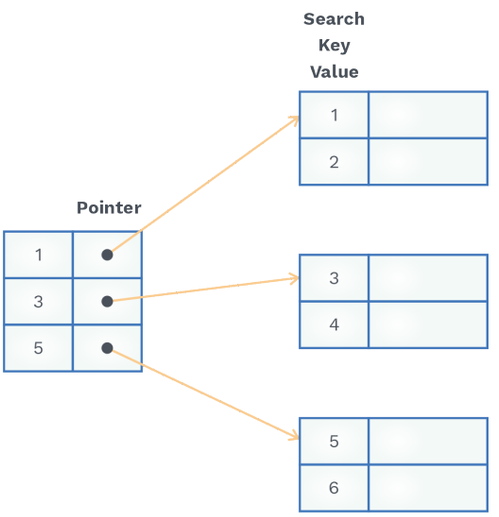


Fig. 5.4 Spare Index

- As we can see in Fig. 5.1, for every search key, there exists an entry in the index file. Thus it is classified as dense index.
- But, in Fig. 5.2, index entries are not available for every search key value. For every disk block, there is subsists an index entry, thus, it is classified as sparse Index.

SOLVED EXAMPLES

Q2 **Consider an ordered file that contains 30,000 records. These records are stored in a disk. The Size of a block is 1024 bytes. The length of file records are is 100 bytes and records cannot be spanned. A primary index structure is employed that contains 9 bytes key and 6 bytes block pointer. Estimate the mean number of block access needed without indexing and with indexing?**

Sol: **Given, size of the block = 1024 bytes**

Size of each record = 100 bytes

Number of records per block $= \frac{1024}{100} = 10.24$

Since records are stored in unspanned manner, we can store only 10 records at maximum.

Number of blocks required to store 30000 records $= \frac{30000}{10} = 3000$

Without primary index

Number of block accesses $= \lceil \log_2 3000 \rceil = 12$

Size of a index record = 9 bytes + 6 bytes = 15 bytes

Number of index record per block $= \frac{1024}{15} = 68$

As we got the number of blocks that is are required to store all these records are= 3000

So, the total number of index records = 3000

$\because$ 68 index records are present in 1 block.

3000 index records will be present in 3000/68 = 45 block.

With primary index

Total number of block accesses $= \lceil \log_2 45 \rceil + 1$

$= 6 + 1 = 7$

Drawbacks of primary indexes:

- The main drawback of a primary index is insertion of records and deletion of records.
- Movement of records is needed to put the new records in the right place.
- But, this will also lead to changes of some of in the index entries.
- **Reason:** Due to the movement of records, it is a high possibility that anchor-records of some of the blocks might get changed.

Definitions

- "Clustering index is created on data file whose file records are physically ordered on a non key field which may not be unique for every record."

2) Clustering indexes:

- A clustering index is an ordered file with two fields:
  1st field contains the data type that is similar to the clustering field of actual file where data is present.
  2nd field contains a block-pointer.

Note:

Clustering Index: An example of a non-dense index.
In clustering indexing, there is an a unique value for every record in the file.

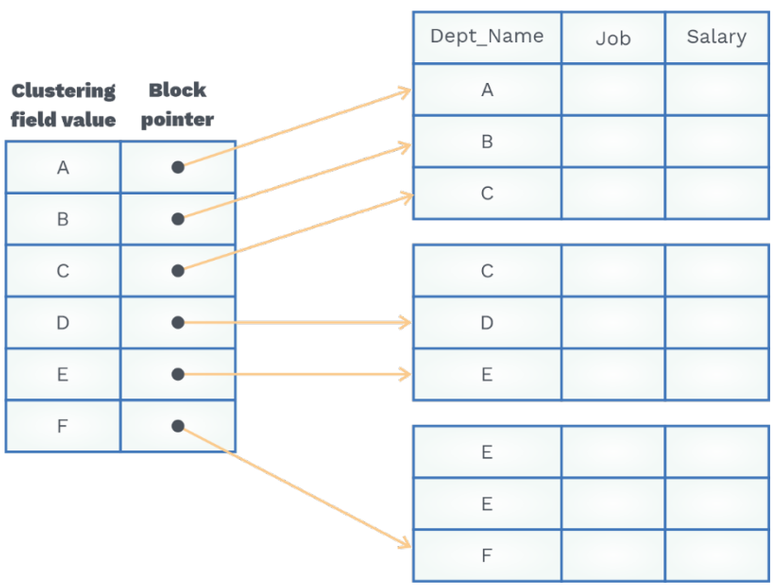


Fig. 5.5 Diagram Represents Clustering Indexing

- The number of block access using clustering index ≥ $\lceil \log_2 Bi \rceil + 1$. where Bi refers to the total block count.

3) Secondary indexes:

Definitions

- A secondary index gives a secondary way to access a data file where primary index or clustering index are already defined.
- Data file records might be ordered, hashed, or unordered.

Note:

The secondary index is based on an unordered field that is
1) CK(Candidate key).
(OR)
2) Non-key field with duplicate values.

The secondary index: ordered file having two fields:
The data type of the first field is same as that of the clustering field of the actual database file.
The second field captures the block pointer.

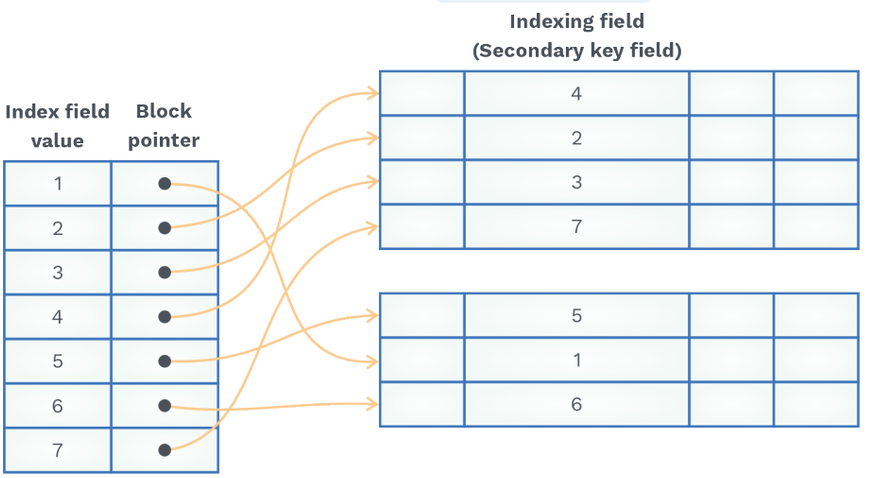


Fig. 5.6 A Dense Secondary Index on a Non-Ordering Key Field of a File

Note:

Number of index entries at first level in secondary index = number of records in the database file.

Previous Years' Question

A file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index. Then that index is called.

1) Dense
2) Sparse
3) Clustered
4) Unclustered

Sol: Option 3) **(GATE-2015 Set-1)**

SOLVED EXAMPLES

Q3 **Assume a database file with 45,000 tuples. This file is organized block-wise in hard-disk. A secondary index structure is used to access the file with key size of 7 bytes. If the sizes of the block, block pointer and record are 2048, 7 and 100 bytes, respectively. If the database follows an unspanned organization, estimate the feasible count of block accesses with and without utilizing index structure.**

Sol: **Block size = 1024 bytes**

Number of data records per block $= \frac{2048\ B}{100\ B} = 20.48$

We can put a maximum of 20 records only in 1 block.

Number of blocks required by data file $= \frac{45000}{20} = 2250$ blocks

Number of index records = number of data records = 45000
Size of index record = 7 + 7 = 14 bytes.

Number of index records per block $= \left\lfloor \frac{2048}{14} \right\rfloor = 146$ entries per block

45,000 entries with 146 entries/block: $= \frac{45,000}{146} = 308$ blocks for the index.

So, without using secondary indexing, we will perform linear search on the file that is equal to $\frac{b}{2} = \frac{2250}{2} = 1125$ block accesses on the average.

With indexing:
A binary search on this secondary index needs $\lceil \log_2 308 \rceil$ = 6 block accesses.

We need one more block-access for searching a record if secondary indexing is used.
Therefore, Total block-access=
6 block access + 1 extra block access = 7 block accesses.

5.3 MULTILEVEL INDEX

- A binary search takes log2bi block access for an index having bi blocks.
  Reason: In each step, part of the index file will reduce by a factor of 2.
- That is why we use a log to the base 2 as a function.
- Due to multilevel indexing, searching becomes much faster.
  Do you know why we need multilevel indexing?
- Sometimes, we have a larger database, and we cannot fit indexes in a single block of a disk.
- Therefore, we need multilevel indexing.
- In multilevel indexing, in order to access a disk block, firstly outer level
- index block is accessed, and a particular entry in the outer level index points to a block in the inner level index block which will point to the disk block which containing the required record.
- y I/O cost to access a record using multilevel indexing having K levels = K+1.

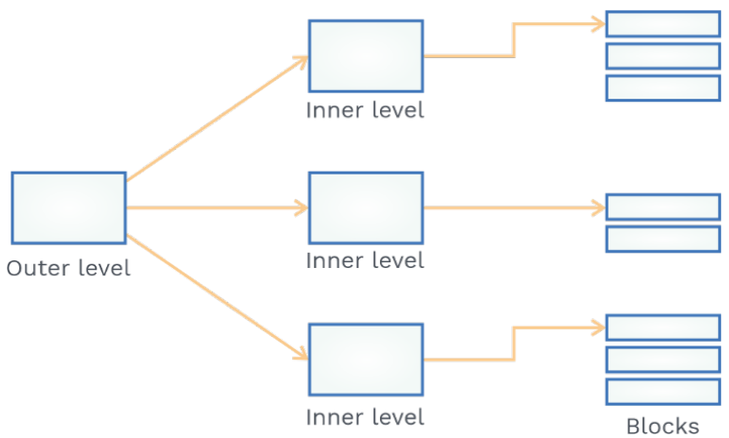


Fig. 5.7

B Tree and B⁺ tree:

- B tree and B⁺ trees are special case of well-known search data structure trees.

Search trees:

Definitions

A search tree is used to search if a record exists or not in the relation based on a given set of field values.

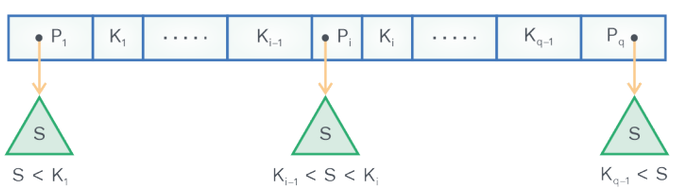
- Suppose P is the order of the Search tree. Then:
- **1)** Maximum search key a node can contain=(P-1)
- **2)** Maximum block pointers a node can contain=P
- **Structure:** $<P_1, K_1, P_2, K_2, ..., P_{m-1}, K_{m-1}, P_m>$ where $m \leq P$ (order of tree).
- Where, P_i is a pointer pointing to a child node and Ki is a search key value.
- **Application:** The Search tree is useful for searching records that are present within a disk-file.
- There are two constraints that need to be followed by a search tree.
  - **i)** Every node should contain the search key value such that, $K_1 < K_2 < ... < K_{m-1}$.
  - **ii)** For all values S in the subtree pointed by P_i,

Grey Matter Alert!

Do you know anything about the TREE data structure?

1) "A tree is defined recursively as a collection of nodes."
2) A root node is present in a tree.
3) Except root node, other nodes are having 1 parent.
4) Also, other nodes can have either no child or more than 0 children. Leaf nodes: It is a node with no children.
5) Internal Node : all nodes other than leaf nodes.

We have:



Grey Matter Alert!

A Tree contains the values from one of the file fields, known as search field.

Example:

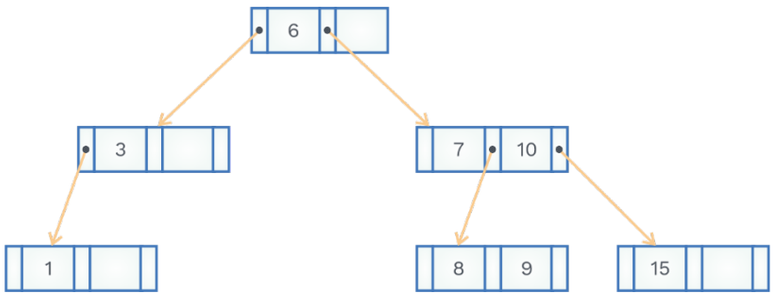


Fig. 5.8 Search Tree of Order P = 3

5.4 B-TREES

- There is an auxiliary condition in B-tree which confirms that the tree is always balanced.
- In B-trees, at every level, we are going to have key and data pointer pointing to either block or record.
- A B-tree of order P can be defined as

  i) Internal node of the B tree is as follows:

$$<p_1, <k_1, pr_1>, p_2, <k_2, pr_2>, \ldots <k_{m-1}, pr_{m-1}>, p_m>$$

where $m \leq p$ (order of tree).

1) Here, each p_i is a tree pointer, and each pr_i is a data pointer.

Tree pointer (Block pointer): A pointer to another node in the B-tree.

Data pointer: It is a pointer that points to the record containing Ki as the search key value.

2) For each internal node, $k_1 < k_2 < k_3 \ldots < k_{m-1}$

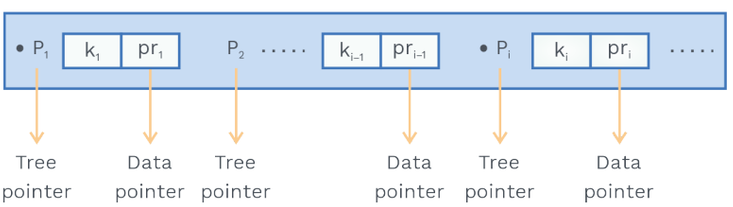
3) Presume S is a search key value in the subtree pointed by p_i. Then:

$K_{i-1} < X < k_i$ for $1 < i < m$

$X < k_i$ for $i = 1$

$K_{i-1} < X$ for $i = m$

4) Maximum count of tree pointers possible is P for each node.



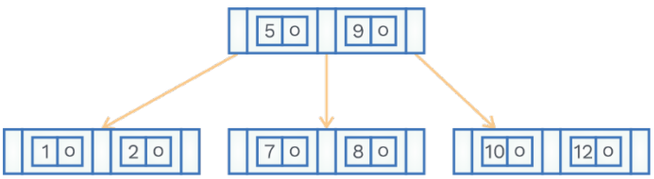


Fig. 5.9 A B-Tree of Prder = 3

B-tree properties:

1) Root node: A root node can have minimum 2 children (Block pointer) and maximum P children.

Where P → order of tree.

Note:

Order of tree: The Order of a tree represents the maximum number of block pointers a node can have. (default definition)

2) Internal nodes:

The Internal node has atleast $\lceil (P/2) \rceil$ tree pointers (Block pointer). It means the internal node can have children between $\lceil P/2 \rceil$ and p.

3) Leaf nodes:

- All leaf nodes are at the same level.
- Also leaf node have the same structure as internal nodes except that all of their pointers p_i are NULL.

Note:

A node with m tree pointers, where m ≤ p(order of the tree) has m–1 data (record) pointers as well as m-1 Search key items(values).

Example: Consider a B-tree of order p = 5. Then the minimum and the maximum number of keys for root, internal and leaf nodes are as follows:

Node	Minimum number of keys	Maximum number of keys
Root	1	4
Internal node	$\lceil 5/2 \rceil - 1 = 2$	p – 1 = 4
Leaf node	$\lceil 5/2 \rceil - 1 = 2$	p – 1 = 4

SOLVED EXAMPLES

Q4 **Consider a B-tree with given data: Size of search key value = 10 bytes, block size of 512 bytes, block pointer is of 5 bytes, and the data pointer is of 8 bytes. What will be the order of the B-tree?**

Sol:

Let n is the order of the B-tree
k represents the key size
p_r represents record pointer
p_b represents block pointer
BS represents block size
A/Q, Given: k = 10 bytes, BS = 512 bytes, p_r = 8 bytes, p_b = 5 bytes.

$n \times p_b + (n - 1) \times (k + p_r) \leq$ Block size

$\Rightarrow \quad n \times 5 + (n - 1) \times (10 + 8) \leq 512$

$\Rightarrow \quad 5n + 18n - 18 \leq 512$

$\Rightarrow \quad 23n - 18 \leq 512$

$\Rightarrow \quad n \leq \frac{530}{23}$

$n \leq 23.04$

$\therefore \quad n = 23$

Thus, the order of B-tree = n = 23.

Q5 **What will be the minimum and maximum number of keys in the B-tree of order 23 for internal nodes?**

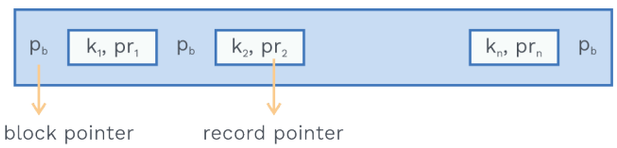
Sol: **For internal nodes,**

Minimum number of keys = $\lceil p/2 \rceil - 1 = \lceil 23/2 \rceil - 1 = 11$

Maximum number of keys = p – 1 = 23 – 1 = 22

Q6 **Consider a B-tree with a search key size of 9 bytes, block size of 512 bytes, record pointer is of 7 bytes, and block pointer is 6 bytes. What is the order of B-tree?**

Sol:



Given, k = key size = 9 bytes, p_r = record pointer = 7 bytes
p_b = block pointer = 6 bytes, BS = block size = 512 bytes
Now, Let 'n' is the order of B-tree

$n(p_b) + (n - 1)(k + p_r) \leq 512$

$\Rightarrow \quad n(6) + (n - 1)(9 + 7) \leq 512$

$\Rightarrow \quad 6n + 16n - 16 \leq 512$

$\Rightarrow \quad 22n \leq 512 + 16$

$\Rightarrow \quad n \leq \frac{528}{22}$

$n \leq 24$

$\therefore \quad n$ = order of B-tree = 24.

Q7 **Suppose order of B-tree is 23. Then how many maximum index records are stored in 4 levels [including root as 1 level] across the B-tree?**

Sol: **Given, The order of the B-tree is 23.**

In level 1: There will be 23 pointers which are equal to the number of block pointer.

Level 1 contains a maximum of 22 index records.

In level 2: In level 1, there are 23 children, and each contains 22 index records at maximum. Thus level 2 will contain at maximum (23) (22) index records.

Similarly, level 3: (23) (23) (22) index records at maximum

level 4: (23) (23) (23) (22) index records at maximum

Thus, overall the maximum index records stored in 4 levels (including root as 1-level) in given B-tree of order 23 = 22 + 23 × 22 + 23 × 23 × 22 + 23 × 23 × 23 × 22

= 22 (1 + 23 + 23 × 23 + 23 × 23 × 23)

= 22 (1 + 23 (1 + 23 + 23 × 23))

= 22 (1 + 23 (1 + 23 (1 + 23)))

= 22 (1 + 23 (553))

= 279840

Searching algorithm in B-tree:

Searching a B-tree is similar to a binary search tree, However rather than moving left or right at each node, we need to perform a p-way search to see which subtree to probe.

Consider an B-tree of order-4.

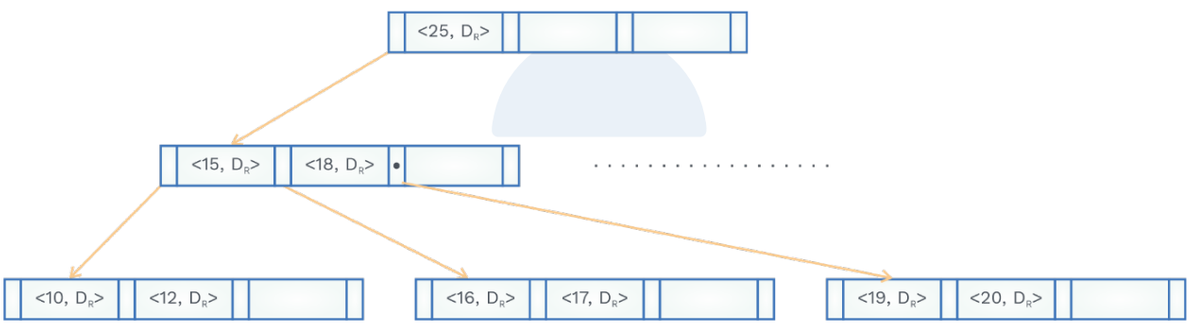


Fig. 5.10 Partially Drawn B-tree of Order 4.

Algorithm:

- Consider a key value K that needs to be searched.
- Searching is done from the root and then we traverse down recursively.
- If K is smaller than the root value, goto left subtree; if K is greater than the root value, search the right subtree.
- If K is found in the node, directly return the node.
- If the node does not contain K, then traverse down to the child with a greater key.
- if K is not found in the tree, return NULL.

E.g. Suppose we wish to search for index record with key value is 17.

Search will start from the root. 17 is not present here and K = 17 is less than root value = 25. We will search in left search tree. Still, K = 17 was not found. Traverse down, K = 17 is greater than value = 15. Thus, search in the right subtree of value 15. Here in the right subtree, we finally found K = 17. Thus, we will return the node.

Time complexity:
Time Complexity of the searching in B-tree = $\log_p n$ where p = order of B-tree and n = number of the search key.

Underflow and overflow in B-trees:

Underflow: If the node has too few values, i.e. has less value than what it is actually required, then this situation is known as underflow.
For roots: If number of search key values < 1., then underflow occurs.
For internal nodes: If the number of search key values < $\left\lceil \frac{P}{2} \right\rceil - 1$ then underflow occurs.

Overflow: Consider a B-tree of order p, if number of search key values in a B-tree node exceeds 'p-1', then this condition is known as overflow.
This is valid for both root nodes as well as internal nodes.

E.g.

Consider the above B-tree of order 5. If we will try to insert new node 9 then overflow occurs.

Rack Your Brain

Consider a B-tree with a search key size of 12 bytes, block size of 1 KB, data pointer is of 5 bytes and blocks pointer is of 8 bytes. Calculate the order of the B-tree?

1) 40 **2)** 41
3) 42 **4)** 43

Insertion in B-tree:
1) At level 0, a B-TREE will contain only one node known as the root node.
2) When level-0 is full, and we have to insert more entries, then the root node will split. We will get one more level as level-1.

Steps:
1) Search to determine which leaf node will hold a key.
2) If the leaf node have space, insert the key in ascending order.
3) Otherwise, split leaf node's keys into two parts and promote median key to the parent.

4) If the parent node is full, recursively split and put the median key to its parent.
5) If a promotion is made to a full root node, split and create a new root having only the promoted the median key.

SOLVED EXAMPLES

Q8 **Consider the B-tree of order 4; and insert A, B, C, D, E, F, G, H, I, J.**

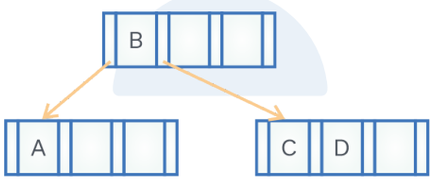
Sol: **As given, the order of B-tree =** 4.

Thus, maximum number of keys = 3.

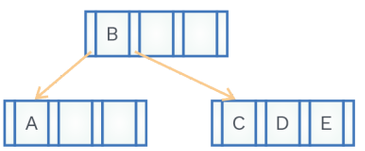
The minimum number of keys = $\left\lceil \frac{4}{2} \right\rceil - 1 = 1$

Insert A, B, C: 

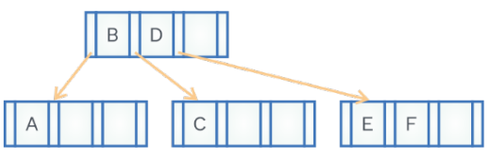
Insert D: When we try to insert D, overflow occurs. Split the leaf node's into two parts and promote the median key to the parent.



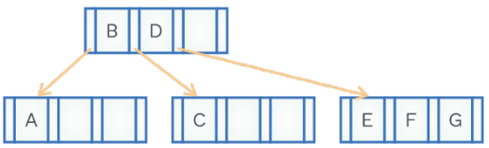
Insert E: When we try to insert E, there will be no overflow. So, just insert E.



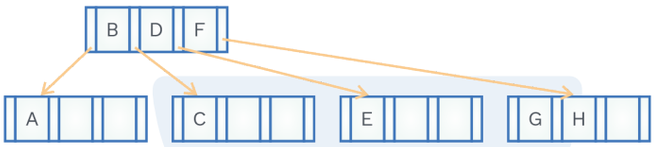
Insert F: When we will insert F into B-tree, then overflow will occur. Split the leaf node's into two parts and promote median keys to the parent.



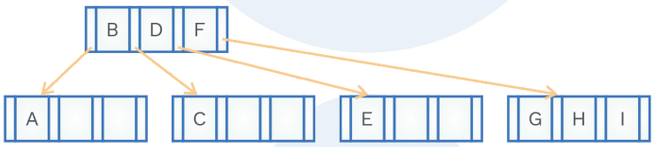
Insert G: When we will insert G, there will be no overflow as space is already available.



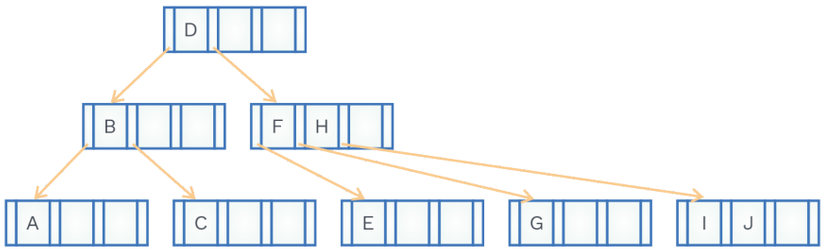
Insert H: When we will insert H into B-tree, then overflow will occur, split the leaf node into two parts and promote the median key to the parent.



Insert I: When we will insert I into B-tree, there will be no overflow.



Insert J: When we will insert J into B-tree, then overflow will occur, split the leaf node into two parts and promote the median key to the parent. Thus now, parent node is also full, so split this node also and promote the median key D to above the level.



Q9 **Consider the B-tree of order 4.**
Insert 2, 5, 10, 11, 1, 6, 9, 4, 3, 12, 18, 20, 25 into B-tree.

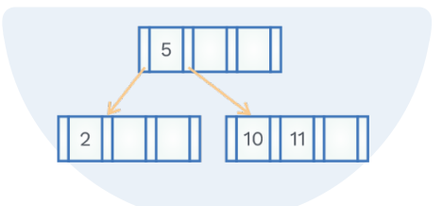
Sol: **As given, the order of B-tree = 4.**

Thus, the minimum number of keys = $\left\lceil \frac{4}{2} \right\rceil - 1 = 1$

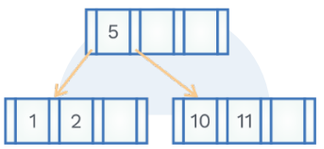
maximum number of keys = 4 – 1 = 3

Inert 2, 5, 10: 

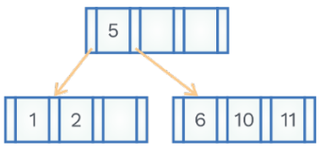
Insert 11: When we try to insert 11, there will be an overflow. Overflow splits the leaf node's into two parts and promotes the median key to the parent.



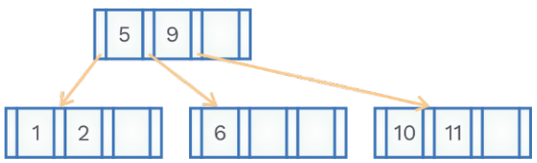
Insert 1: When we try to insert 1 into B-tree, there will be no overflow.



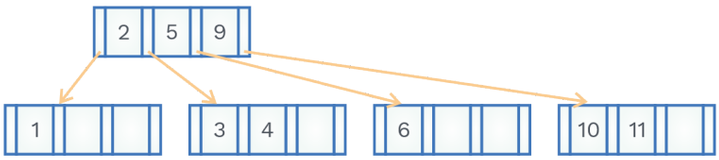
Insert 6: When we try to insert 6 into B-tree, there will be no overflow.



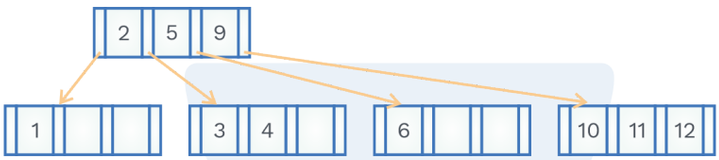
Insert 9: Leaf node will have overflow now. So split the leaf nodes into two parts and promote median key to the parent.



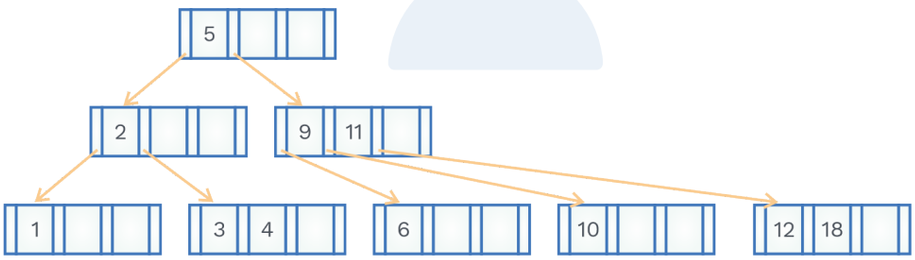
Insert 4 and 3: When we insert 4, there will be no overflow, but when we insert 3 there will be overflow and thus split the leaf node and into two parts and promote the median key, which is k = 2 to parent.



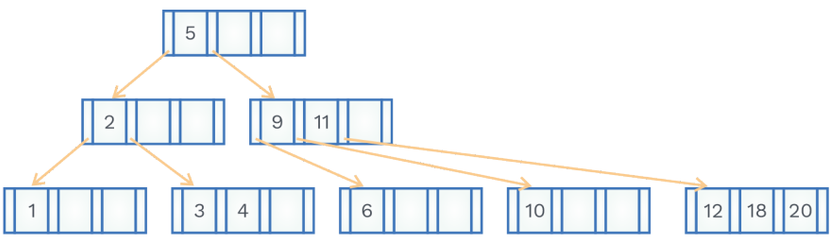
Insert 12: When we try to insert 12 then there will be no overflow.



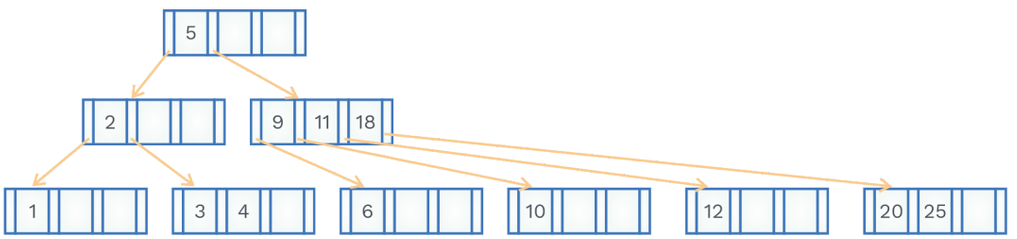
Insert 18: Now leaf node will have overflow; thus, we will split leaf node and promote 11 to the upper (parent) level.
Further, the parent level will also get overflowed. So, split nodes and promote 5 to upper level. This is shown below.



Inset 20: When we try to insert 20 into B-tree then, there will be no overflow



Insert 25: When we will insert 25, the leaf node will have an overflow, so split the leaf node and promote key value 18 to parent node.



Previous Years' Question

A B-tree used as an index for a large database table has four levels, including the root node. If a new key is inserted in this index, then the maximum number of nodes that could be newly created in the process are

1) 5 **2)** 4 **3)** 3 **4)** 2

Sol: Option 1) **(GATE-IT-2005)**

Deletion from B-tree:

Steps:

1) If a non-leaf key value is to be deleted, interchange it with its successor or predecessor and delete it from its new leaf node position.

2) If a leaf node block stores more than the minimum possible count of keys, simply delete the targeted key from the leaf node block. Again, if there exists a minimum count of keys in a leaf node block, the two immediate sibling leaf nodes are to be considered as described in further points.

3) If any sibling leaf node contains more than the minimum permissible count of keys, transfer the median key value to the parent node and a key from the parent to the node with deficit of keys.

4) If both of them contains exactly the minimum allowed count of keys, the deficit node is merged along with one of the siblings and a parent key.

5) If the above steps result in to less number of keys in parent node, repeat all the steps in an upward manner.

6) If this leaves the parent node with too few keys, then the process is propagated upward.

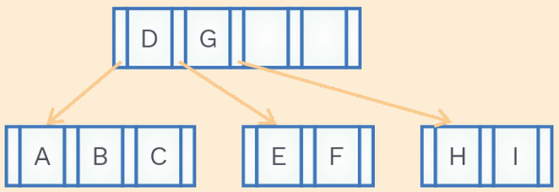
Let removing a key from a leaf node leaves l-keys in the leaf node.

i) If $l > \left\lceil p/2 \right\rceil - 1$, then we can stop, i.e. no underflow.

ii) If $l < \lceil p/2 \rceil - 1$, then this is a condition for underflow and we must rebalance the tree.

SOLVED EXAMPLES

Q10 **Consider the B-tree of order 5 given below:**

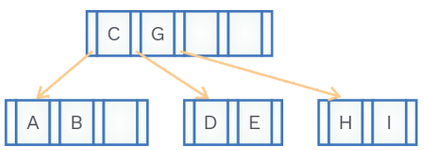


Delete F and D from the B-tree.

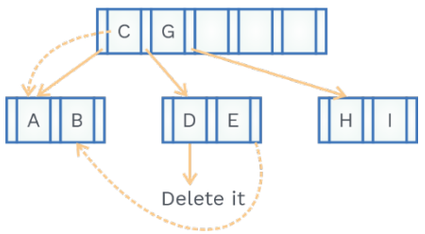
Sol: **Since the order of B-tree = 5.**

So, the maximum number of keys in leaf/internal node = 4 and the minimum number of keys in leaf/internal node = $\lceil p/2 \rceil - 1 = 2$.

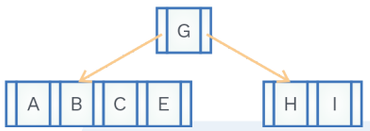
Delete F: Borrow Method; since F is present in the leaf node and the minimum number of keys which should be present in leaf node is 2. Thus, when we delete node F, we need to rebalance the B-tree. For that, we will Borrow a node from one of the siblings if the siblings have more than a minimum number of keys. Redistribute one key from this sibling to the parent node and one key from the parent to the deficient node.



ii) Delete D: Coalesce → merging method.
D is present in the leaf node, and the minimum number of keys which should present in the leaf node is 2.
Thus when we delete node D from the leaf; and both immediate siblings have exactly the minimum number of keys, we will merge deficient node with one of the sibling node, an one entry from the parent node.



⇒



5.5 B+ TREES

A variation of B tree that is more efficient in searching for an element.

Hey learners!!

- Do you know why we use the B+ tree?
- In the B tree, the number of entries that can be present within a node is less as the structure of the B tree contains a data pointer along with the search key.
- B+ tree eliminates this drawback.
- In the B+ tree, the data pointer can be stored only at the leaf nodes.
- Thus, the structure of the leaf node and internal node both are different.
- In the B+ tree, all the key values must be present in the leaf node.
- One leaf node is linked to the other leaf node so that we can access the search key in an ordered way.
- There are some repeated search field values present in leaf node that is also present in internal nodes of the B+ tree.

Rack Your Brain

Consider a B tree having search key 12 bytes long, data pointer B bytes long, block pointer 4 bytes long, and block size of 1024 bytes. What will be the maximum number of children a node can have _____?

- **Internal node:** Consider P= order of a B+Tree.
  Structure:
  1) Internal node can be represented as :
  $<P_1, K_1, P_2, K_2, \ldots P_{m-1;}\ K_{m-1}, P_m>$ where each P_i is tree pointer.
  2) Each internal node must have, $K_1 < K_2 < K_3 \ldots < K_{m-1}$
  3) For every internal nodes, maximum P tree pointer is possible.
  4) Each internal node except the root has atleast $\lceil P/2 \rceil$ tree pointers.
  5) Internal node has $\lceil P/2 \rceil - 1$ to P – 1 search key values.
- **Root node:** The root node contains atleast 2 tree pointers.
- **Leaf node:** Structure of the leaf nodes of a B^+ tree of order P:
  $<<K_1, Pr_1>, <K_2, Pr_2>, \ldots, <K_{m-1}, Pr_{m-1}>, P_{next}>$
  where $m \leq P$ (order of tree), Pr_i = record/data pointer and P_{next} = pointer pointing to the next leaf node of B^+ - tree.

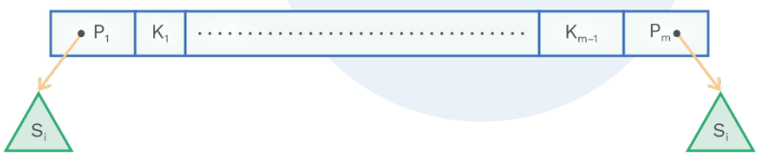


Fig. 5.11 Non-leaf Structure of a B^+ Tree

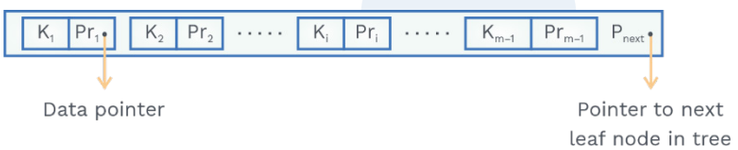


Fig. 5.12 Leaf Node Structure of a B^+ Tree

- $K_1 \leq K_2 \leq K_3 \ldots\ldots \leq K_{m-1}$ where $m \leq p$ (order of tree)
  [within each leaf node]
- Each leaf node has at least $\lceil P/2 \rceil$ values.
- All leaf nodes need to be at the same level

Note:

An internal node of a B+ tree contains more entries than a B tree.

Note:

As we got to know that the structure of leaf nodes and internal nodes are different for a B+ tree, therefore order can also be different.

- P is the order of internal nodes.
- P_{leaf} is the order for leaf nodes.
- Order of leaf represents maximum number of data pointers in a leaf node.
- Order of non-leaf (internal) node represents maximum number of children a node can have.

SOLVED EXAMPLES

Q11 **Consider the data given below for the B+ Tree:**
i) Size of the Search key field is 9 bytes long
ii) Size of block = 512 bytes
iii) Size of record pointer = 7 bytes
iv) Size of block pointer = 6 bytes
What will be the order of the leaf node and internal node?

Sol: **For internal node:**

As we know, an internal node of the B+ tree has pointers = P and search key value =(P-1)

Thus, $P * (\text{block pointer}) + (P - 1) * (\text{key field}) \leq \text{Block Size}$

$\Rightarrow P * (P_b) + (P - 1)(V) \leq \text{block size}$

$\Rightarrow P * (6) + (P - 1) * (9) \leq 512$

$\Rightarrow 15\ P \leq 521$

$\Rightarrow P = 34$

$\Rightarrow$ $O_{non-leaf}$ = Order of non-leaf (internal node)
P = 34

Structure of Leaf node

$(K_1, Pr_1)\ (K_2, Pr_2)\ \ldots\ldots\ (K_m, Pr_m)\ P_b$

$P_{leaf}\ (K + Pr) + P_b \leq \text{Block size}$

Where K = search key field
Pr = record pointer
Pb = block pointer

$\Rightarrow \quad P_{leaf}(9 + 7) + 6 \leq 512$

$\Rightarrow \quad P_{leaf} * 16 \leq 512$

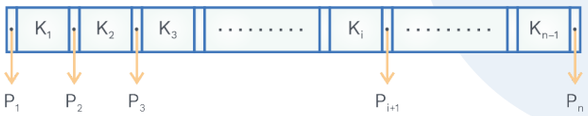
$\Rightarrow \quad P_{leaf} \leq \frac{506}{16}$

$\boxed{P_{leaf} = \text{order of leaf} = 31}$

Searching a record with key value K in B⁺ tree of order P

Algorithm:

- Start the search from the root, look for the largest key value K_i in the node, which is less than equal to key value K.
- Follow the pointer P_{i+1} to the next value until reach the leaf node.



- If K is found to be equal to K_i in the leaf, then follow Pr_i to search record.
- **Insertion in B⁺ - tree:**
- Overflow condition in B⁺ tree: When the number of search key values exceed"P-1" then this condition is overflow in B⁺ - tree.
- **Case 1:** Overflow in a leaf node, then,
- Split the leaf node into 2 nodes.
- First node will contain $\left\lceil \frac{(P-1)}{2} \right\rceil$ values where $\lceil\ \rceil$ denotes ceil function.
- Second node will contain the remaining values.
- Copy the smallest search key value of the second node to the parent node.
- **Case 2:** Overflow in non-leaf node then,
- Split the non-leaf node into two nodes.
- First node will contain $\left\lceil \frac{P}{2} \right\rceil - 1$ values/ keys.
- Move the smallest of the remaining keys to the parent.

Previous Years' Question

The order of a leaf node in a B⁺ tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?

1) 63 **2)** 64
3) 67 **4)** 68

Sol: Option 1) **(GATE-2007)**

Rack Your Brain

Which of the following are correct about B-tree and B⁺ tree?

S1: In a B-tree, every value of the search field appears at once at some level in the tree.

S2: An internal node in B⁺ tree with n pointers have (n + 1) search key field values.

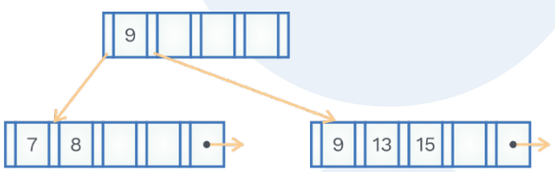
1) Only S1
2) Only S2
3) Both S1 and S2
4) Neither S1 nor S2

- Second node will contain remaining keys.

Example: Consider the B^+ tree of order 5.

Insert 8

- As the order of B^+ tree = 5. Thus, the maximum number of search keys values present inside a leaf = 4.
- When we try to insert K = 8 in the given B^+ tree, overflow occurs.
- Split the leaf nodes such that the first node contains $\left\lceil (P-1)/2 \right\rceil = 2$ key value (i.e. 7 and 8).
- Second node will contain the remaining key values (i.e. 9, 13, and 15).
- Smallest search key value of the second node will be copied to the parent node.



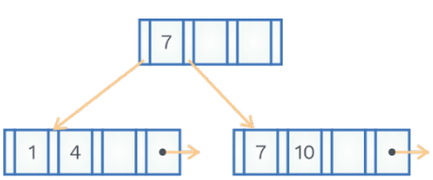
SOLVED EXAMPLES

Q12 **Construct a B^+ tree for the insertion sequence 1, 4, 7, 10, 17, 21, 31, 25 with P = 4 and P_{leaf} = 4**

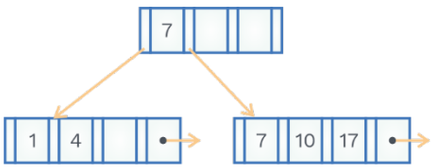
Sol: **As P = 4 and P_{leaf} = 4 is given. It means maximum number of key value present in either leaf or non-leaf node is P – 1 = 3.**

1) Insert 1, 4, 7

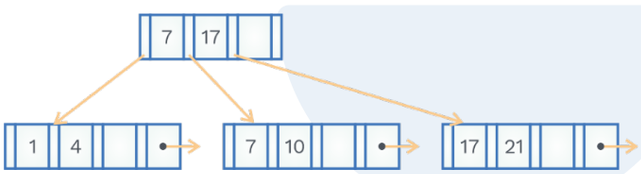
2) Insert 10 (overflow) ; $\left\lceil (P-1)/2 \right\rceil = 2$



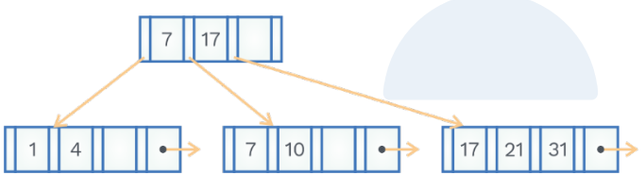
3) Insert 17 (No overflow): Insert it into the right leaf.



4) Insert 21 (overflow) calculate $\lceil (P-1)/2 \rceil$:2 split the leaf node such that the first node will contain 2 elements, and the remaining elements will be present in other node.

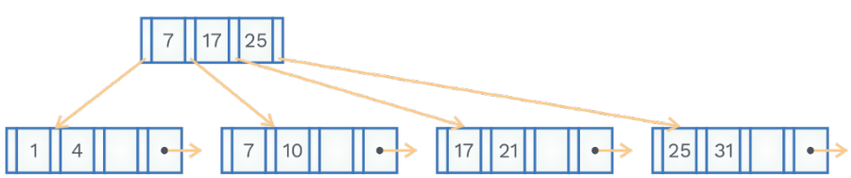


5) Insert 31 (no overflow):



6) Inert 25 (overflow): Split the leaf node such that the first node will contain

key values (i.e. 17 and 21), and the second node will contain the remaining key values (i.e. 25 and 31), and the smallest search key value i.e 25 will be copied to the parent node.



Deletion of an entry from B^+ trees:

- When we have to delete any entry from B^+ trees, first it will be deleted from the leaf node.
- If the same entry is also present in an internal node, it is required to remove that entry from here also.
- Deletion leads to the underflow situation as it will decrease the number of entries less than the required one in the leaf node.

Previous Years' Question

In a B^+ tree, if the search key value is 8 bytes long, the block size is 512 bytes and the pointer size is 2B, then the maximum order of the B^+ tree is _____

Sol: Range: 52 to 52.

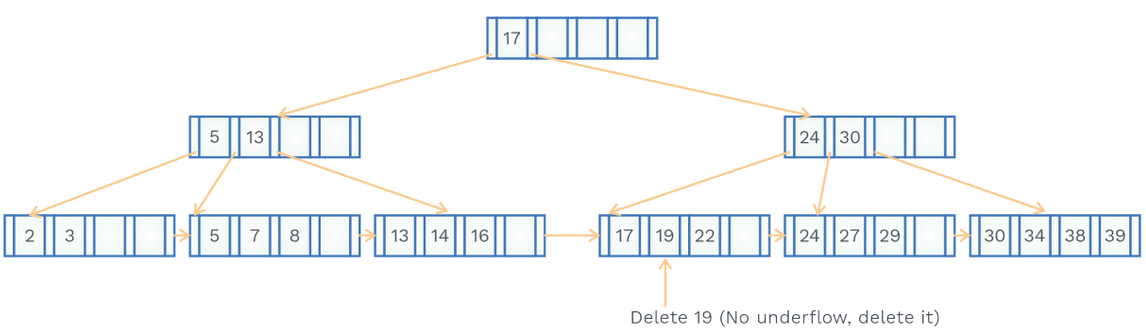
(GATE-2017 Set-2)

Steps:

- Start at root, find leaf L where entry belongs.
- Remove the entry
  - **i)** If L contains at least $\lceil P/2 \rceil - 1$ entries, done.
  - **ii)** If L has only $\lceil P/2 \rceil - 2$ entries.
    - **1)** Redistribution of entries means take borrow from the adjacent node whose parents are the same as L (also known as a sibling).
    - **2)** If redistributing fails, merge L and a sibling.
- If a merge has occurred, then corresponding entry from parent must be deleted.
- Merge could propagate to root, decreasing height.
- If the deleted entry is present in internal node, replace it with inorder successor.

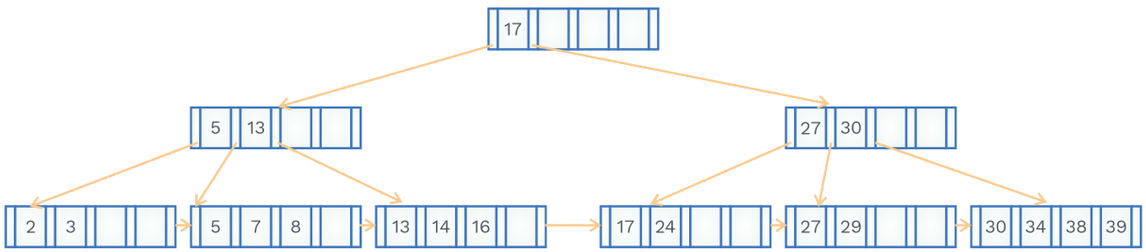
Example: Consider the B^+ tree of order P = 5. Deletion sequence: 19, 22

1) Delete 19:



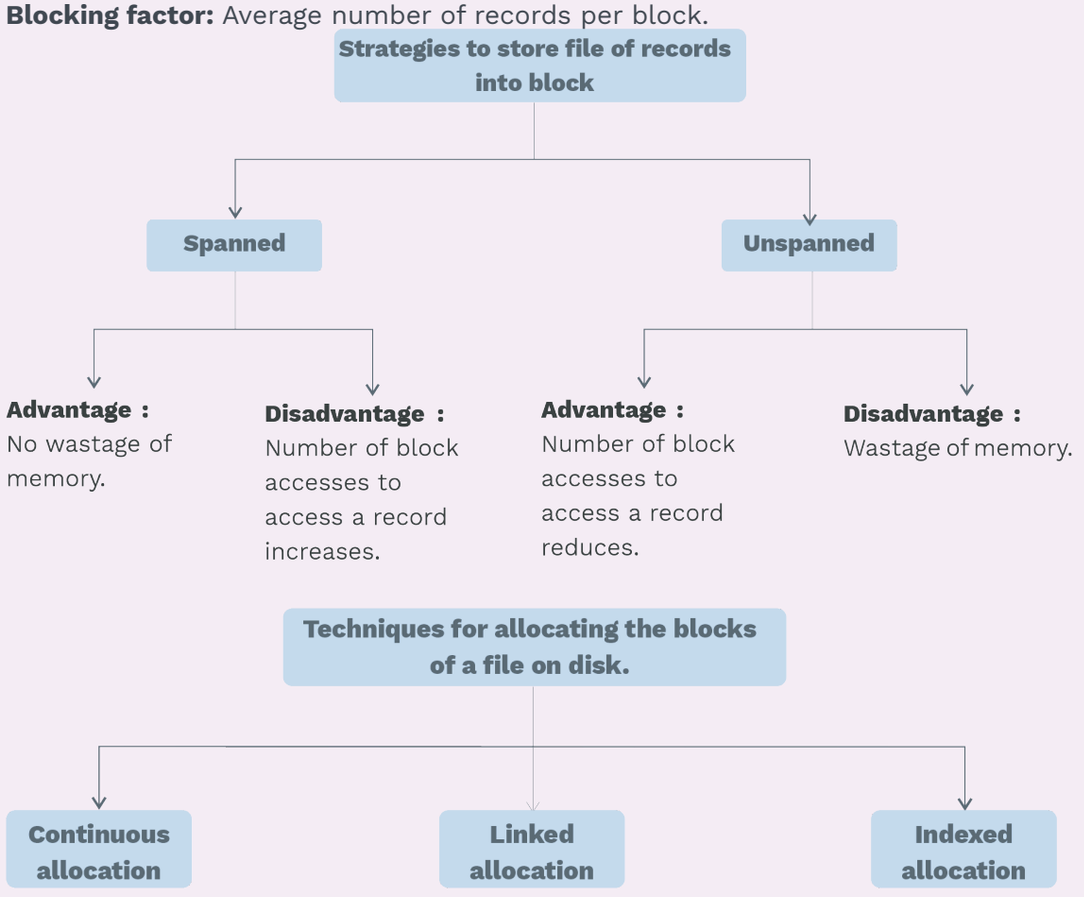
2) Delete 22:

Deleting 22 leads to underflow as $\left\lceil \frac{P}{2} \right\rceil - 1 = 2$ but leaf contains < 2 entries after deleting 22. Thus, redistribute borrowing from a sibling.



Chapter Summary

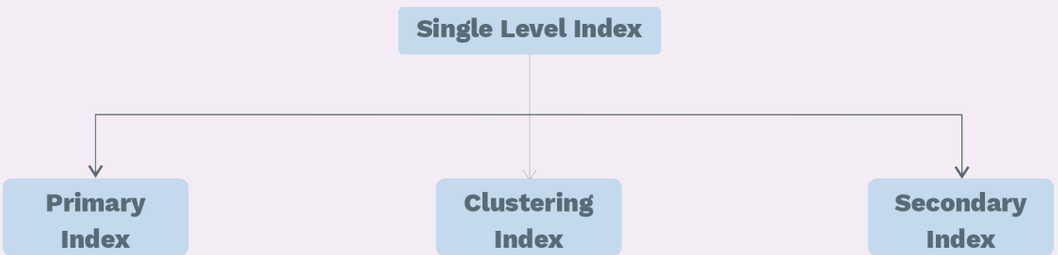
- **Files:** It is organised as a sequence of records.
- **Unordered file:** Records are placed in no particular order.
- **Blocking factor:** Average number of records per block.



- **File organization:** It refers to the organization of the data of a file into records, blocks and access structure

	Types of Organization	Access/Search Method	Average blocks to access a specific record
i)	Heap (unordered)	Linear Search (Sequential scan)	b/2
ii)	Ordered	Linear Search (Sequential scan)	b/2
iii)	Ordered	Binary Search	$\log_2 b$

- There are types of indices based on ordered files (single-level-indexes) and tree data structures (multilevel indexes, B^+ trees).



- **Anchor record:** The anchor record is the first record in each block of the data file.
- **Dense index:** If an index entry is created for every search key value, then that index is called the dense index.
- **Sparse index:** If an index is created only for some search key value, then that index is called the sparse index.
- The number of blocks accesses using clustering index $\geq \lceil \log_2 B_i \rceil + 1$ where Bi refers to the total block count.
- The secondary index may be created on an unordered field that is the non-key field or candidate key.
- Number of index entries in secondary index = number of records.
- **Multilevel index:** It is used to speed up the search operations.
- **Search tree:** It contains maximum (P-1) keys, P pointers if P is defined as order of a search tree.
- B-tree and B^+ trees are special cases of search data-structure trees.
- **B-tree:** A B-tree of order P can be defined as:

Each internal node in the B-tree is of the form

$$<P_1, <K_1, Pr_1>, P_2, <K_2, Pr_2>, \ldots\ldots <K_{m-1}, Pr_{m-1}>, P_m>$$

Where m ≤ P (order of tree), Pi is the tree pointer, and Pri is a data pointer.

- All leaf nodes in B-tree are at the same level.
- **Order of tree:** the order of the tree represents the maximum number of children (Block pointer) a node can have.
- Time complexity of searching in B-tree = logpn where p = order of B-tree and n = number of the search key.