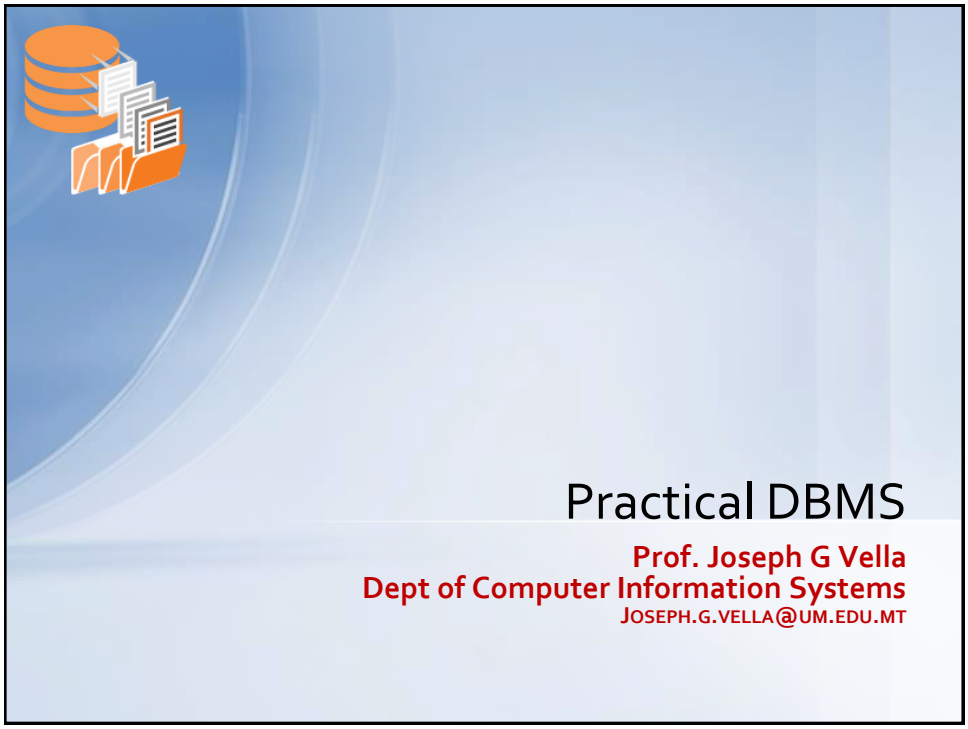

Practical DBMS
Prof. Joseph G Vella
Dept of Computer Information Systems
JOSEPH.G.VELLA@UM.EDU.MT

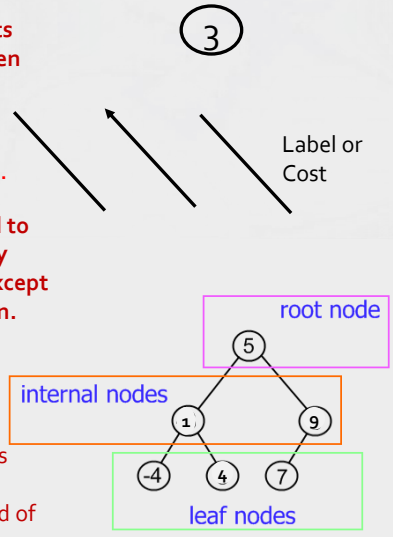
1


Practical DBMS
Indexing: B Tree

2

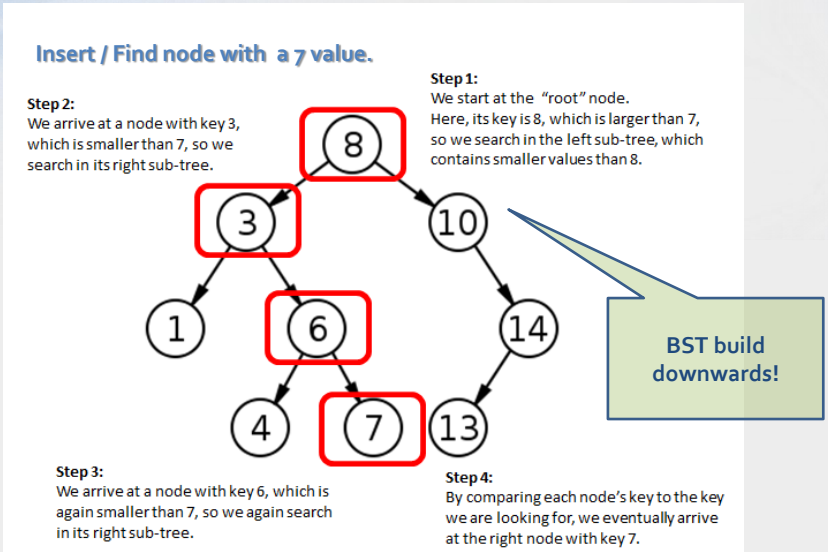
Tree Basics

- A tree is a data structure whose components are a set of nodes and a set of edges between the nodes.
  - Each node has value(s);
  - Each edge connects two nodes;
    - edges could be directed and labeled too.
- A binary tree has a directed edges (i.e. child to parent – 5 is parent of 3), each child has only one parent (except root), and each node (except leaf nodes) is a parent of one or two children.
- Example:
  - A root has no parent;
  - A leaf has no kids;
  - If node 5 is level 0 then 3 is level 1, and 7 is level 2;
  - A sub-tree rooted at value -8 is composed of nodes -8, -4, and 9.

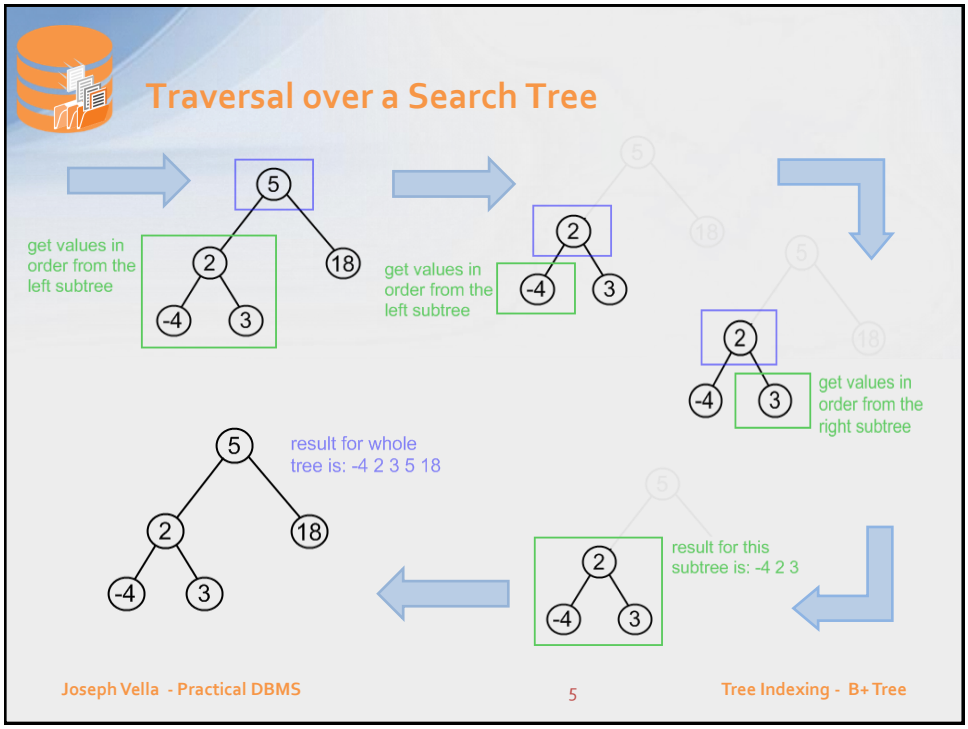


3

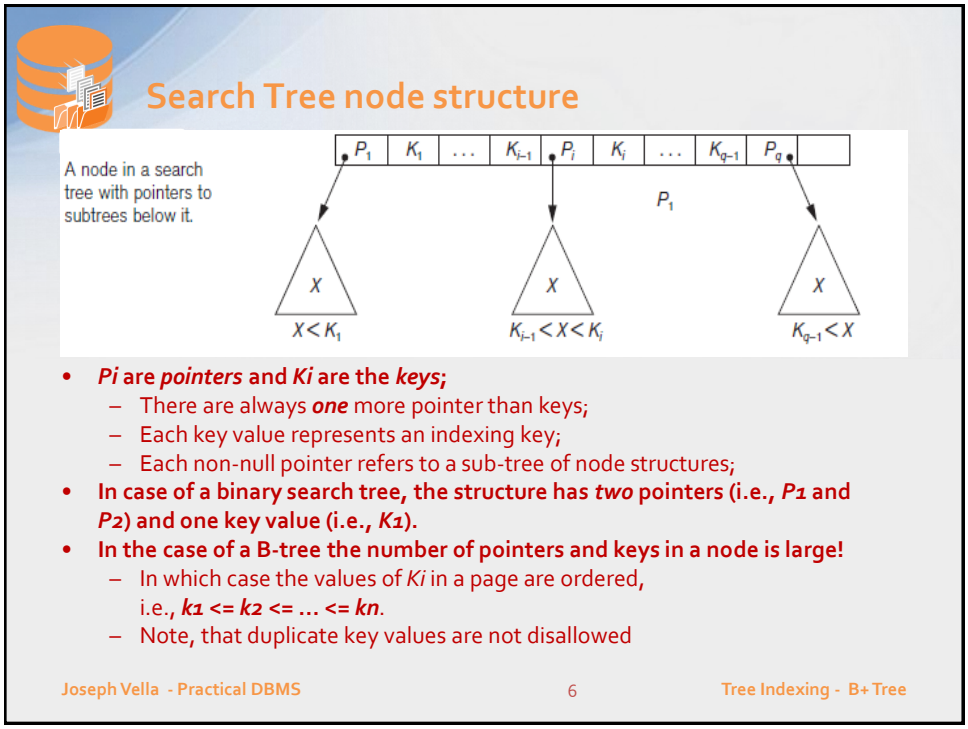
Tree Searching (and Building): Effective only possible if placement is ordered



4

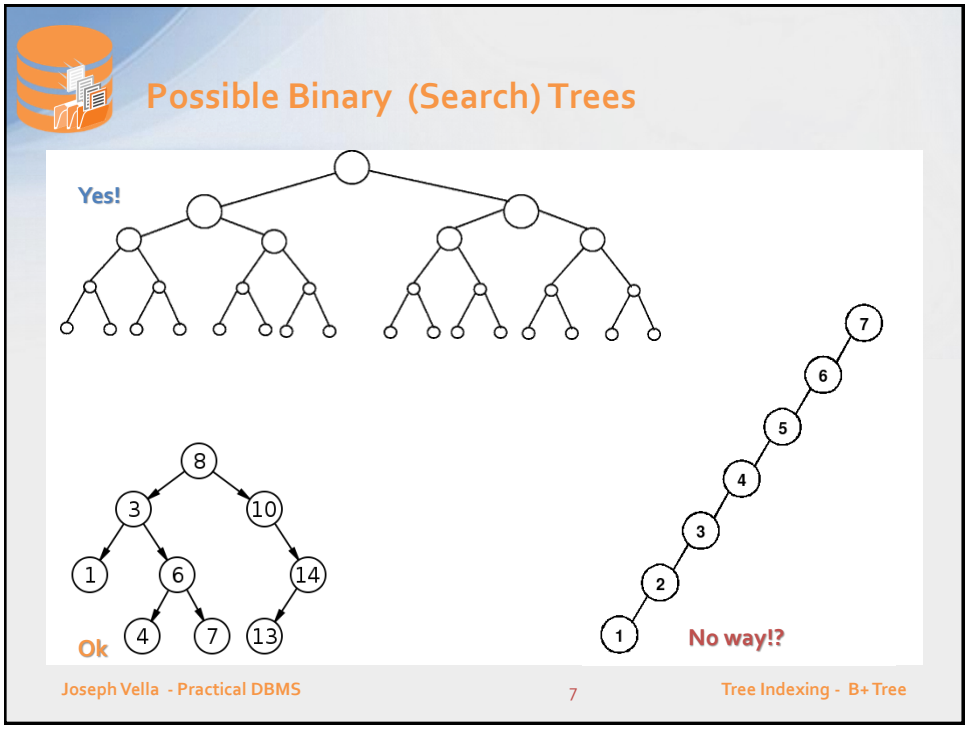


5

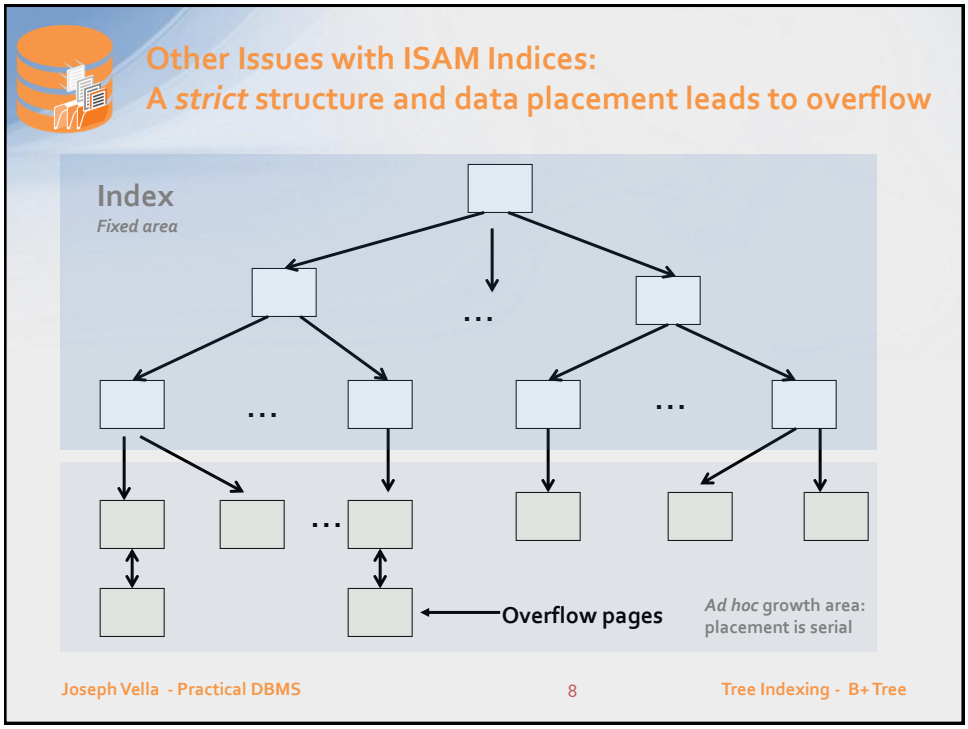


- ***Pi*** **are** ***pointers*** **and** ***Ki*** **are the** ***keys*;**
  - There are always ***one*** more pointer than keys;
  - Each key value represents an indexing key;
  - Each non-null pointer refers to a sub-tree of node structures;
- **In case of a binary search tree, the structure has *two* pointers (i.e., *P1* and *P2*) and one key value (i.e., *K1*).**
- **In the case of a B-tree the number of pointers and keys in a node is large!**
  - In which case the values of *Ki* in a page are ordered, i.e., ***k1 <= k2 <= ... <= kn***.
  - Note, that duplicate key values are not disallowed

6


Possible Binary (Search) Trees
Yes!
8
3
10
1
6
14
4
7
13
Ok
7
6
5
4
3
2
1
No way!?
Joseph Vella - Practical DBMS
7
Tree Indexing - B+ Tree

7


Other Issues with ISAM Indices:
A strict structure and data placement leads to overflow
Index
Fixed area
Overflow pages
Ad hoc growth area:
placement is serial
Joseph Vella - Practical DBMS
8
Tree Indexing - B+ Tree

8

Requirements (or desiderata) and Issues

- Tree structure is to be balanced;
  - And *bushy*;
- Tree structure has all leafs at the same level;
- Block based;
- Allows direct and sequential reads;
- All activities have a similar cost basis;
- High space utilisation (per block);
- Resilient to some abuse;
- Sharing mechanism is possible and reasonable.

9

B-Tree Motivation and Approach

- The key technique required to move away from binary search trees (and their variants) to a new solution, B-trees, in that we can choose to ***build trees upwards from the bottom instead of downwards from the top***.
  - We have seen how binary trees are stuck with the root node!
- ***Bayer and McCreight 1972*** (at that time employed by Boeing's but "B" is for Bayer the supervisor) recognized that rather than keeping a bad root and having ever more complex re-arrangement algorithms one should avoid having the root node "anchored down" at the first place!
  - ***The B-trees allows the better root to emerge from the key's present while keeping the structure balanced.***

10

B-Tree Characteristics

- **B-trees genera of data structures lend themselves perfectly to paging!**
  - Performance is further enhanced when the root page (and the higher levels perhaps) is kept in ultra fast storage memory (for example in a RAM cache or very fast hard disk).
- **B-trees computational requirements are more on the deterministic side:**
  - can work out the index size and number of levels by knowing the number of keys and page size (and hence order);
  - can work out the range of min and max "time" required to compute operations over B-tree index data structure.
    - **Caution: some of the algorithms look complicated, and indeed they are, but their computational costs is not related to algorithms' structure.**
- **B-trees have been in use since 1972.**
  Informatics have not been the same since then!?

11

B-Tree: Basic Definitions

- **Basic definitions:**
  - ***order***
    - **is the maximum number of descendants a node/page can have.**
      - for example, if a page can hold at most seven keys and eight descendants and a minimum of three keys and four descendent then its *order* is eight.
  - ***root***
    - ***It's the first page to read and where the first mutli-way split – the higher the order the more partitions possible.***
      - ***Assuming that the root's key are *good* partitions.***
  - ***leaves***
    - **are the pages at the lowest level of the B-tree; they do not have any descendants.**

12

B-Tree: Simple example (order = 3)
Search keys are unique

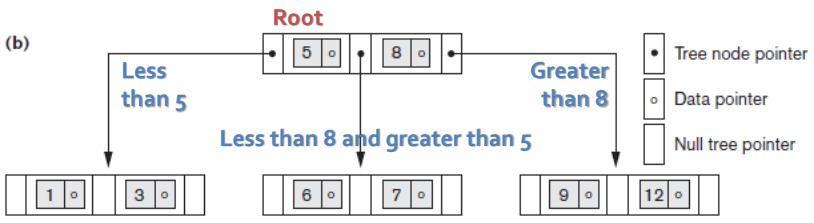
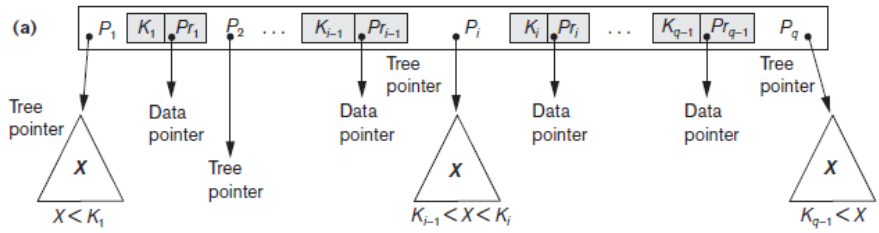


Figure 18.10
B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

13

B-Tree: Page Structure



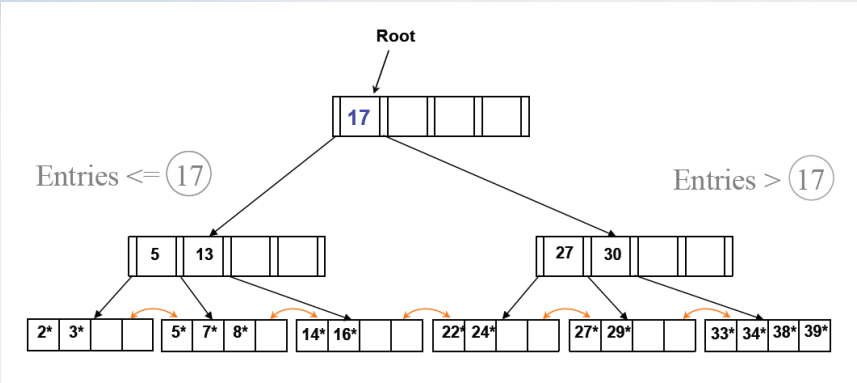
k1 <= k2 <= ... <= kn

Data Structure

```
TYPE
   BTPAGE     =
   RECORD
     KEYCOUNT :  integer;
     KEY      :  array[1.. MAXKEYS] of char;
     CHILD    :  array[1.. MAXCHILDREN] of integer
   END;
```


14

B-Tree: Point and Range Example Queries



B-Tree Properties

- **B-Tree: properties for an *m*-order tree**
  1) Every page has a maximum of ***m*** descendants;
  2) The root has at least two descendants (unless it is a leaf);
  3) All the leaves appear on the same level;
  4) A nonleaf page with ***k*** descendants contains ***(k-1)*** keys;
  5) A leaf page contains at least ***(ceiling(m/2)-1)*** keys and no more than ***(m-1)*** keys.
  6) Every page except for the root and the leaves has at least ***ceiling(m/2)***;
  7) Any keys present in a page are in order (i.e. $k_1 <= k_2 <= \ldots <= k_n$).
  8) For all search keys, whose value is X and in a sub-tree pointed by *Pi* we have:
     Ki-1 < X Ki for 1 < I < q
     X < Ki for i=1
     Ki-1 < X for i=q

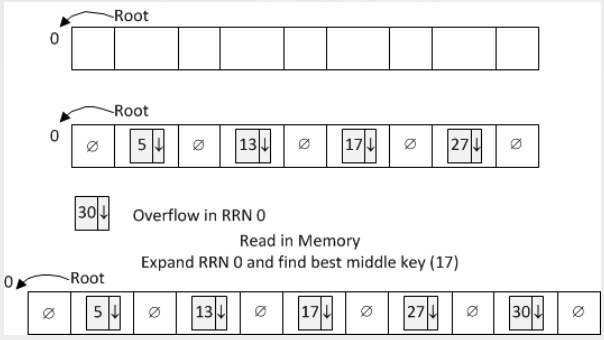
B-Trees Deliver

- B-Trees are balanced;
- B-Trees are shallow (distance from root node to any leaf page is short) and consequently require less disk reads;
- Random insertions and deletions in B-Trees are accommodated at a reasonable and predictable costs;
- At least loads 50% use of claimed space - empirical tests put the loading average value at 66%;
- Sequential searches (across pages) not implicit but possible with some effort!
  - *look up some details on B+.*
- B-Trees are flexible to – can accommodate non-key search field values – we need to change the definition of pointer Pri.

17

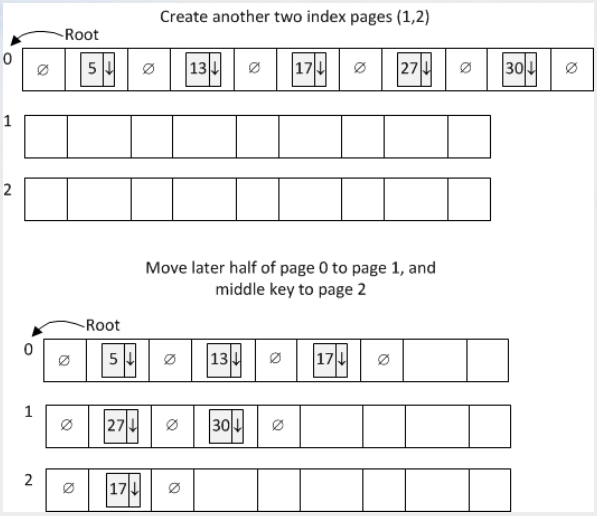
B-Tree: Basic Operation – Split & Promote

- B-Tree with order set to 5 (therefore 4 index entries)
- Assume the following key order:
  - 27 [RRNi] , 17 […], 13 […], 5 […], 30 […]



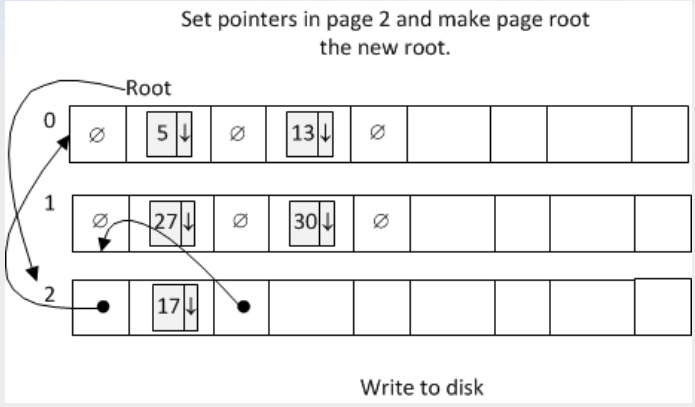
18

B-Tree: Basic Operation – Split & Promote

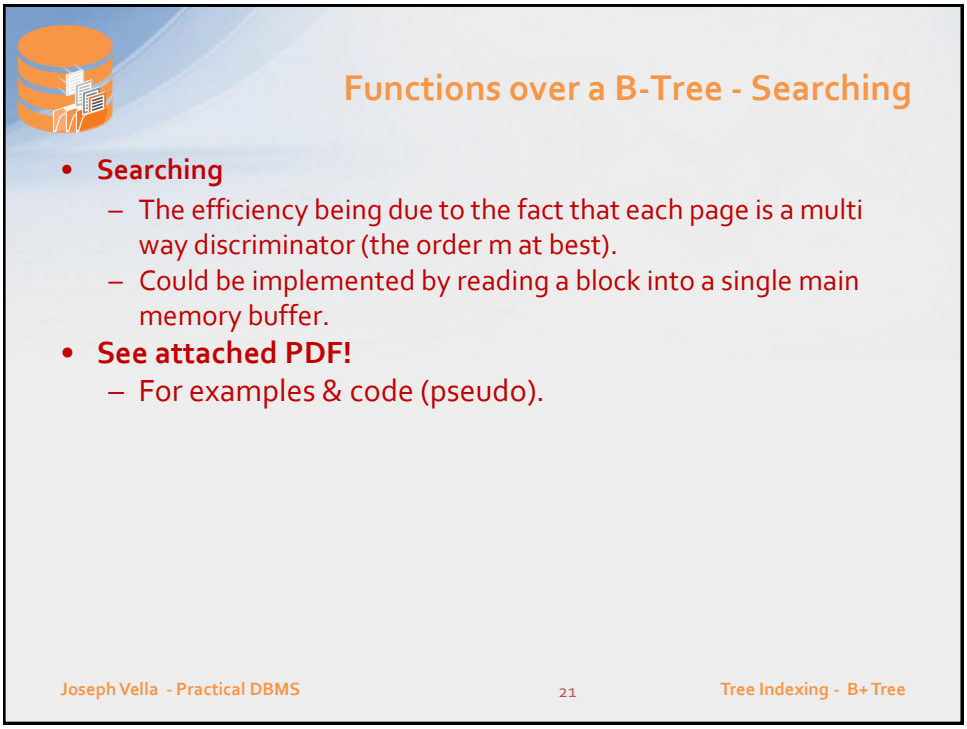


19

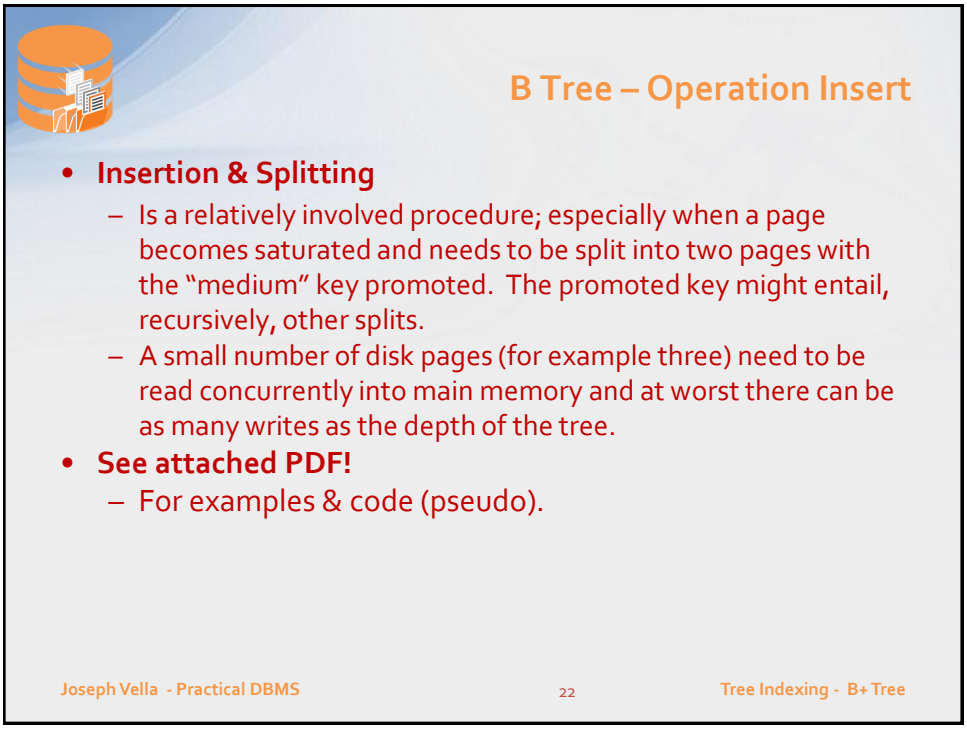
B-Tree: Basic Operation – Split & Promote



20


Functions over a B-Tree - Searching
• Searching
– The efficiency being due to the fact that each page is a multi way discriminator (the order m at best).
– Could be implemented by reading a block into a single main memory buffer.
• See attached PDF!
– For examples & code (pseudo).
Joseph Vella - Practical DBMS
21
Tree Indexing - B+ Tree

21


B Tree – Operation Insert
• Insertion & Splitting
– Is a relatively involved procedure; especially when a page becomes saturated and needs to be split into two pages with the "medium" key promoted. The promoted key might entail, recursively, other splits.
– A small number of disk pages (for example three) need to be read concurrently into main memory and at worst there can be as many writes as the depth of the tree.
• See attached PDF!
– For examples & code (pseudo).
Joseph Vella - Practical DBMS
22
Tree Indexing - B+ Tree

22

B-Tree Operation - Delete

- **Deletion**
  - Definitely the trickiest part of the implementation.
  - If a non leaf key needs to be purged, the next key (by logical order) has to raise and fill the gap - thus a balanced organisation is maintained.
  - Also, if a page's key count is below the acceptable level then re-grouping to the preceding (by logical order) page must be initialled.
  - A small number of pages have to be read concurrently into main memory. The number of writes equal that of inserts.
- **See attached PDF!**
  - For examples & code (pseudo).

23

B-Tree – Cost of Operations

- **All of these operation require time proportional to the depth of the B tree.**
  - That is $O(\log_n m)$.

24

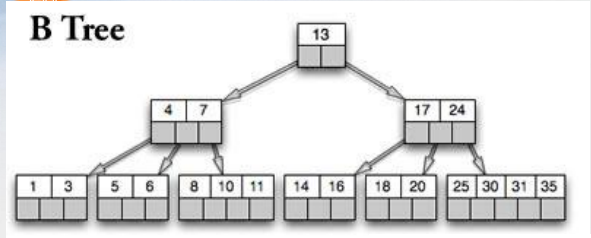
Working out the worst-case search depth

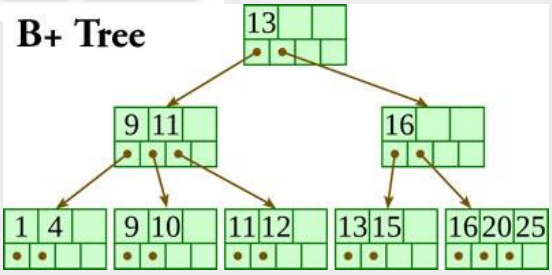
- **Given a B-tree, of order m, and a large number of keys, N, what is the worst-case scenario as regards the tree's depth, d?**
  - For example, say we have a million keys and a 512 ordered key; what will be the max number of page reads required to find a key's entry in the index.
- **Observation One:**
  - The number of descendants a certain tree level has is equal to the number of keys present in that level and all the other higher levels, plus one.
    - **see attached figure!**
- **Observation Two:**
  - What is the min number of descendants that any level can have?
    - **This permutation yields a B tree that has the max height and the min of breath.**
  - Specifically:
    - **the root page has 2 as the min;**
    - **the second level has `ceiling(m/2)` per page (i.e., 2);**
    - **...**
    - **the nth level has `ceiling(m/2)`$^{d-1}$ * 2.**
- **Workings:**
  - $N+1 >= ceiling(m/2)^{d-1} * 2$
  - solving for d;

$$d =< 1 + \log_{ceiling(m/2)}((N+1)/2)$$
$$d =< 1 + \log_{256} 500000.5$$
$$d =< 3.366$$

25

B Trees vs B+ Trees





https://www.differencebetween.info/difference-between-b-tree-and-b-plus-tree

26

PostgreSQL and B+ indices

- **To create an index in PostgreSQL:**
  CREATE TABLE emp(…);
  – Note that an index is built on any PK defined.
  CREATE INDEX idx_emp_job ON emp(job);
  CREATE INDEX idx_emp_deptno_job ON emp(deptno,job);

27

Index Selection Issues

- **Consider emp table (especially attributes empno, sal, job and deptno)**
  – Intense usage of:
  SELECT *
  FROM emp
  WHERE job = $bind_variable;
  – Occasional usage of:
  SELECT *
  FROM emp
  WHERE deptno=$bind_variable;
    - **Which indexes earn their keep?**
      Surely index on job.
      If only it's the query on deptno but there is plenty of data in emp then index on deptno (and job) seem to be useful

28

Index Selection Issues

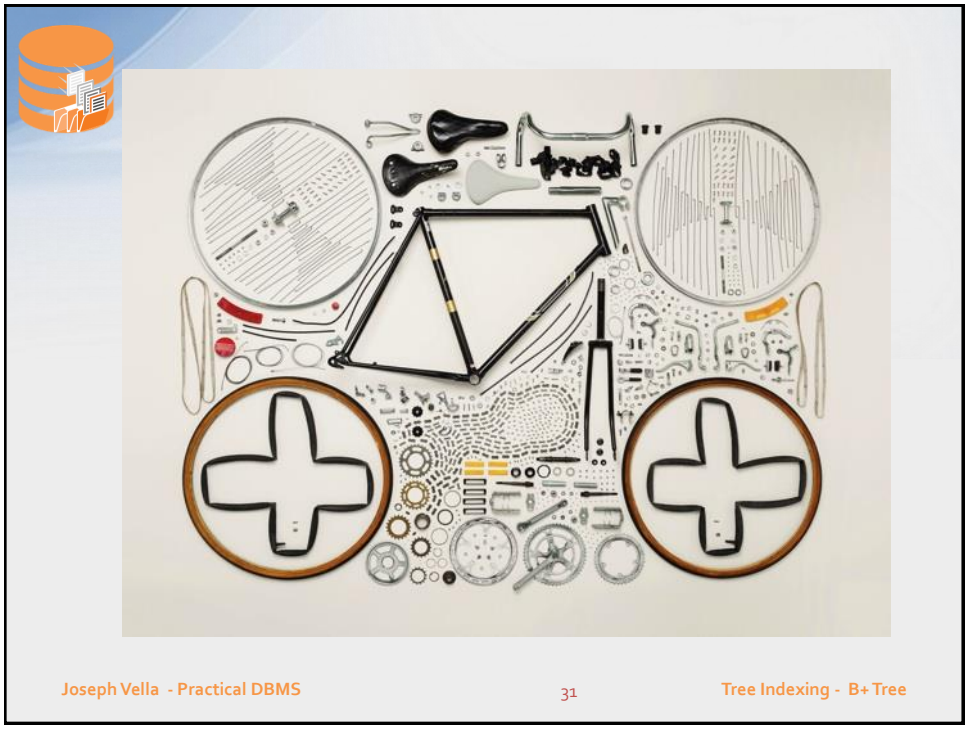
- **Heavy use of the following comparison query:**
  - SELECT *
    FROM emp
    WHERE deptno > $bind_var1
      AND deptno <= $bind_var2;
  - INSERT INTO emp
    VALUES (…);
    - **Index on deptno, job would be very useful (consider the prefix only).**
    - **Every index need updating.**
- **Heavy use of:**
  - SELECT *
    FROM emp
    WHERE deptno = $bind_var1
      AND job = $bind_var2;
    - **Yes …**

29

B-Tree and batch/bulk uploads

- **Case study:**
  - Is it possible?

30



31