5.28 B-Tree Deletion in Data Structures | DSA Tutorials

Implementation of DBMS

Exercise Sheet 9

Klingemann, WS 2024 / 2025

1. **Delete from a B+-tree:**
   Starting with the **B+-tree of order 4** you created in task 1 of Sheet 8, delete the keys 45, 20, 10, 50, 35, 40, 60, 30, 25, 15, 70, 55, 90 in this order. Show the tree structure after each operation and describe any necessary redistributions or merges.
2. **Insert into a B-tree:**
   Insert the keys 100, 50, 25, 125, 75, 150, 30, 60, 40, 80, 120, 135 into an initially empty **B-tree of order 2** and if have time order 3. Show the tree structure after each insertion and explain any splits.
3. **Delete from a B-tree:**
   From the **B-tree of order 2** you produced in task 2, delete the keys 40, 60, 120, 25 in this order. Show the tree structure after each deletion and explain all redistributions or merges that occur.

Implementation of DBMS

Like Exercise Sheet 9

Klingemann, WS 2024 / 2025

1. **Insertions in a B+-Tree**
   Insert the keys 5, 15, 25, 35, 45, 10, 20, 30, 40, 50 into an initially empty **B+-tree of order 3**. After every insertion, show the structure of the tree and any splits that occur.
2. **Deletions in a B+-Tree**
   Starting with the B+-tree of order 3 produced in task 1, delete the keys 20, 40, 10, 25 in this order. Show the tree structure after each deletion and explain any merges or redistributions that occur.
3. **Insertions and Deletions in a B-tree**
   Insert the keys 12, 8, 18, 5, 15, 20, 10 into an initially empty **B-tree of order 4**. Then, delete the keys 15, 20, 8 in this order. Explain all structural changes, such as splits or merges, during both insertion and deletion operations.

Implementation of DBMS

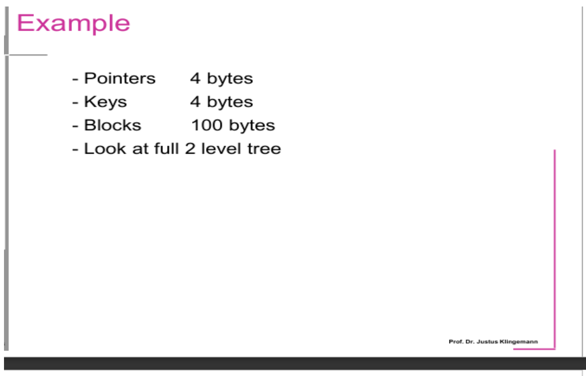
Again, similar to **Exercise Sheet 9**
Klingemann, WS 2024 / 2025

1. **B+-Tree Operations**
   Starting with an initially empty **B+-tree of order 4**, insert the keys 2, 8, 12, 4, 6, 10, 14, 16, 18. After completing all insertions, delete the keys 12, 6, 14 in this order. Show the tree structure after each operation and explain any necessary redistributions or merges.

2. **B-tree Operations**
   Insert the keys 3, 9, 15, 7, 12, 1, 4 into an initially empty **B-tree of order 3**. Show the structure of the tree after all insertions. Then delete the keys 7, 4, 9 in this order and show how the tree changes.

3. **Compare B+-Tree and B-tree**
   Discuss the differences between the B+-tree and the B-tree structures produced in tasks 1 and 2 above. Focus on:

   - Handling of keys during splits and merges.
   - The organization of leaf and non-leaf nodes.
   - Use cases where B+-trees might be preferred over B-trees in database systems.

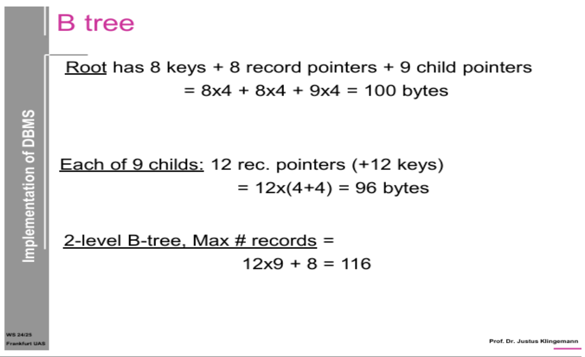


Example

- Pointers	4 bytes
- Keys	4 bytes
- Blocks	100 bytes

- Look at full 2 level tree

Prof. Dr. Justus Klingemann



Implementation of DBMS

B tree

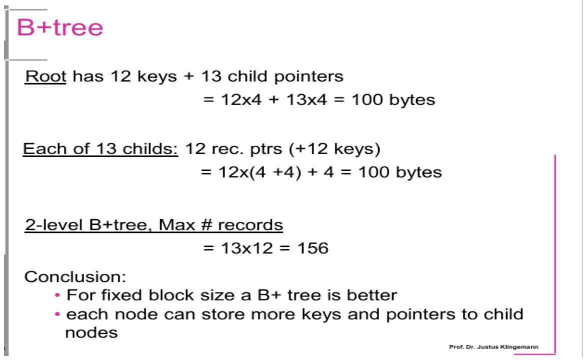
Root has 8 keys + 8 record pointers + 9 child pointers
= 8x4 + 8x4 + 9x4 = 100 bytes

Each of 9 childs: 12 rec. pointers (+12 keys)
= 12x(4+4) = 96 bytes

2-level B-tree, Max # records =
12x9 + 8 = 116

WS 24/25
Frankfurt UAS

Prof. Dr. Justus Klingemann



B+tree

Root has 12 keys + 13 child pointers
= 12x4 + 13x4 = 100 bytes

Each of 13 childs: 12 rec. ptrs (+12 keys)
= 12x(4 +4) + 4 = 100 bytes

2-level B+tree, Max # records
= 13x12 = 156

Conclusion:
- For fixed block size a B+ tree is better
- each node can store more keys and pointers to child nodes

Prof. Dr. Justus Klingemann

Explanation of B-tree and B+ tree Example in the Image

B-tree Example

A **B-tree** is a balanced tree structure used for indexing databases. In the given example:

1. **Root Node:**
   - **8 keys** (each 4 bytes) → $8 \times 4 = 32$ bytes.
   - **8 record pointers** (each 4 bytes) → $8 \times 4 = 32$ bytes.
   - **9 child pointers** (each 4 bytes) → $9 \times 4 = 36$ bytes.
   - **Total Size of Root Node** → $32 + 32 + 36 = 100$ bytes.
2. **Child Nodes (Each of 9 children):**
   - **Each child stores 12 records, with 12 keys.**
   - **Each key is 4 bytes, and each record pointer is 4 bytes.**
   - **Total per child** → $12 \times (4 + 4) = 96$ bytes.
3. **Maximum Records in a 2-level B-tree:**
   - Each of the 9 children holds 12 records → $9 \times 12 = 108$ records.
   - Plus, the 8 records in the root → $108 + 8 = 116$ records.

B+ tree Example

A **B+ tree** is an optimized version of B-tree where internal nodes only store keys, and all records are stored in leaf nodes.

1. **Root Node:**
   - **12 keys (each 4 bytes)** → $12 \times 4 = 48$ bytes.
   - **13 child pointers (each 4 bytes)** → $13 \times 4 = 52$ bytes.
   - **Total Size of Root Node** → $48 + 52 = 100$ bytes.
2. **Child Nodes (Each of 13 children):**
   - **Each child stores 12 records with 12 keys.**
   - **Each key (4 bytes) and record pointer (4 bytes).**
   - **Additional 4 bytes per child.**
   - **Total per child** → $12 \times (4 + 4) + 4 = 100$ bytes.
3. **Maximum Records in a 2-level B+ tree:**
   - Each of the 13 children holds 12 records → $13 \times 12 = 156$ records.

Why Does Each Child Store 12 Records?

- The number of keys per node is determined by the block size and key/pointer size.
- The goal is to **maximize storage within each block** while ensuring efficient searching.
- Given the **100-byte block size**, and each key-pointer pair taking **8 bytes**, we can fit:

$$\frac{100}{8} \approx 12 \text{ records (rounded down)}$$

Conclusion

- The **B+ tree can store more records** in the same block size because **internal nodes only store keys**, allowing **more efficient searches** and **faster range queries**.
- For **fixed block size, B+ trees are preferred** because they store more keys and child pointers.

Three More Exercises

Exercise 1a:

Consider a **B-tree** where:

- **Pointers are 8 bytes, keys are 8 bytes, and blocks are 200 bytes.**
- The **root node** has **10 keys, 10 record pointers, and 11 child pointers**.
- Each **child node** stores **15 records**.

Tasks:

1. Calculate the **size of the root node**.
2. Compute the **size of each child node**.
3. Find the **maximum number of records** in a **2-level B-tree**.

Exercise 1b:

Consider a **B+ tree** where:

- **Pointers are 6 bytes, keys are 6 bytes, and blocks are 150 bytes.**
- The **root node** has **9 keys and 10 child pointers**.
- Each **child node** holds **13 records**.

Tasks:

1. Compute the **size of the root node**.
2. Calculate the **size of each child node**.
3. Find the **maximum number of records** in a **2-level B+ tree**.

Exercise 1c:

A **B-tree variant** has:

- **Pointers: 5 bytes, Keys: 5 bytes, Blocks: 120 bytes**.
- The **root node** contains **7 keys, 7 record pointers, and 8 child pointers**.
- Each **child node** holds **10 records**.

Tasks:

1. Compute the **size of the root node**.
2. Determine the **size of each child node**.
3. Find the **maximum number of records** in a **2-level B-tree**.

Example 1

- **Pointers**: 8 bytes
- **Keys**: 8 bytes
- **Blocks**: 200 bytes
- **Look at full 2-level tree**

Example 2

- **Pointers**: 6 bytes
- **Keys**: 6 bytes
- **Blocks**: 150 bytes
- **Look at full 2-level tree**

Example 3

- **Pointers**: 5 bytes
- **Keys**: 5 bytes
- **Blocks**: 120 bytes
- **Look at full 2-level tree**
