Space and Time Trade-Offs

Jiří Dvorský, Ph.D.

Presentation status to date February 24, 2025

Department of Computer Science
VSB – Technical University of Ostrava

Lecture outline

Space and Time Trade-Offs

B-trees

B-trees – motivation

- Processing a large amount of structured records (that can be identified by a unique key) that exceeds the available operating memory.
- Data must be stored in external memory, so-called ”on disk”.
- The disk offers only a sequential file.
- We are looking for a data structure that allows efficient searching, inserting, and deleting records in such a file.
- The answer is to trade off memory complexity for time complexity, in other words, we increase memory complexity (we sacrifice extra memory) to reduce the time complexity of operations.

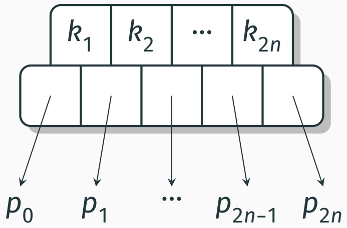
B-trees

B-tree of order n is a $(2n + 1)$-tree that satisfies the following criteria:

1. Each page contains at most $2n$ keys.
2. Each page, with the exception of the root, contains at least n keys.
3. Each page is either a leaf page, i.e. it has no children, or it has $m + 1$ children, where m is the current number of keys in the page.
4. All leaf pages are on the same level. In other words, the tree is perfectly balanced.

Published by Rudolf Bayer in 1972 [1].

B-trees – page schema



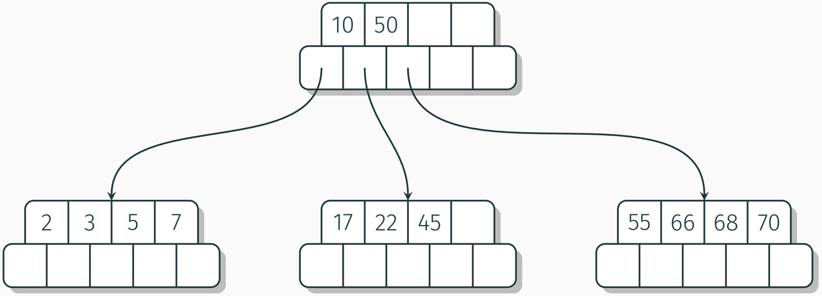
- Nodes in a B-tree are traditionally called **pages**.
- The number of keys in a page ranges from n to $2n$, with the exception of the root node.
- Keys in a page are sorted, i.e., $k_1 \leq k_2 \leq \cdots \leq k_{2n}$.

- For keys in the subtrees referenced by pointers $p_0, \ldots, p_{2n}$, the following holds

$$K_{p_0} \leq k_1 \leq K_{p_1} \leq k_1 \leq \cdots K_{p_{2n-1}} \leq k_{2n} \leq K_{p_{2n}},$$

where K_{p_i} is the set of all keys in the subtree rooted at the page referenced by p_i.

B-trees – example



B-trees – notes

- From the definition, it is clear that a B-tree does not have to be completely filled. The fill factor varies from 50 % to 100 %.
- Free space in the tree allows for easy insertion of additional keys.
- Thanks to the tree structure, search, insert, and delete key operations in a B-tree can be performed with logarithmic time complexity.
- B-tree algorithms are a generalization of binary search tree algorithms.

B-trees – alternative definition, variants

- The above definition only allows B-trees with a maximum capacity of $2n$ keys, i.e., an even number.
- However, the maximum capacity can be any number, even odd.
- Some definitions denote the maximum capacity using the number n.
- The number of keys in a page thus varies from $\left\lceil \frac{n}{2} \right\rceil$ to n.
- The goal of our definition is easy understandability of the B-tree operation principles and simple notation.
- Sometimes, one can also encounter a definition where the number n denotes the maximum number of children, not the number of keys in a page.

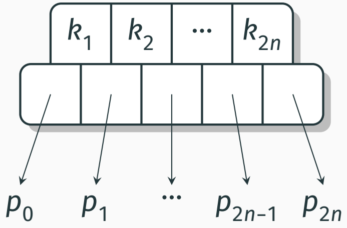
B-trees – variants

B^+-tree	• all keys are stored only in leaves • leaves are mutually linked by pointers – faster operation with contiguous key ranges, ”find all keys between 100 and 200” • in Levitin’s book [2] is as a B-tree described precisely this variant.
B^*-tree	• a page must be filled to at least two thirds, • when inserting a key into a full page, keys are first moved between siblings, • results in a smaller number of page splits.

B-trees – searching for a key x

1. At the beginning of the algorithm, we mark the root page as the current page P.
2. If page P does not exist, the search ends in failure.
3. Otherwise, assume that page P contains m keys $k_1, \dots, k_m$ and corresponding child pointers $p_0, \dots p_m$. Then:
   3.1 If $x = k_i$, for some $1 \le i \le m$, then the search ends in success.
   3.2 If $x < k_1$, then $P = p_0$ and back to point 2.
   3.3 If $x > k_m$, then $P = p_m$ and back to point 2.
   3.4 Otherwise, we find such i, $1 \le i < m$, for which it holds that $k_i < x < k_{i+1}$. Then $P = p_i$ and back to point 2.

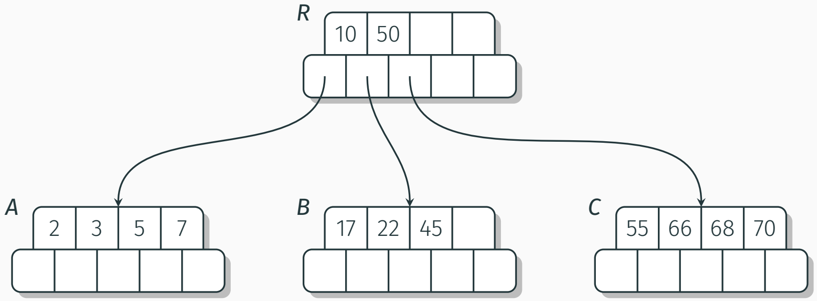
Examples of B-tree operations



In examples we will use B-tree for $n = 2$, meaning each page contains at least 2 and at most 4 keys.

Furthermore, each page refers to at least 3 and at most 5 children. The exception is the root of the tree.

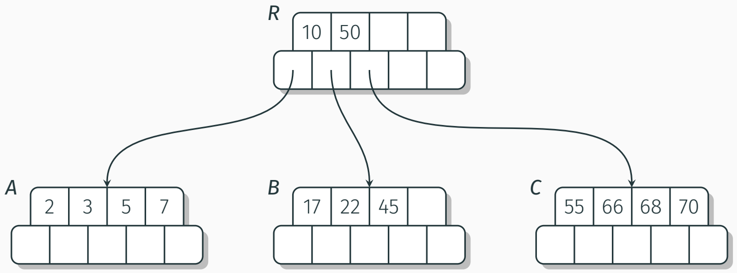
Example of searching in a B-tree – finding the key 50



Procedure

1. We start the search in the root R, so $P = R$.
2. Page P exists, we proceed to the next point.
3. Because $x = k_2$ the search ends in success.

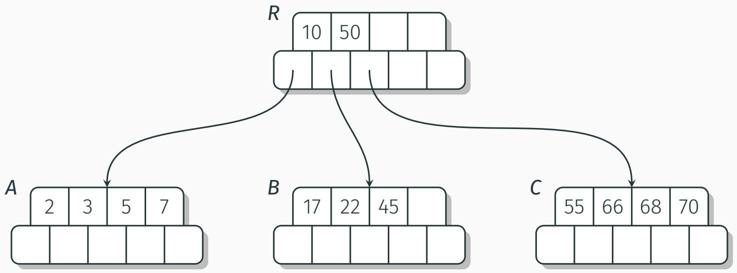
Example of searching in a B-tree – search for 3



Procedure

1. We start the search in the root R, thus $P = R$.
2. Page P exists, we proceed to the next point.
3. Since $x < k_1$, then $P = p_0 = A$.
4. Page P exists, we proceed to the next point.
5. Since $x = k_2$ the search ends in success.

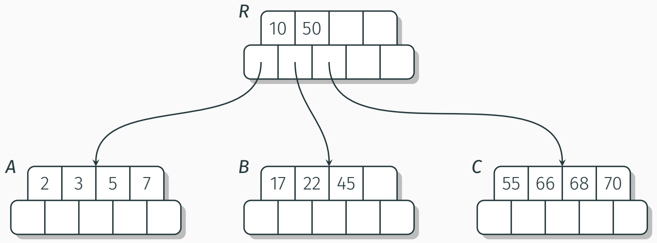
Example of searching in a B-tree – search for 45



Procedure

1. We begin the search at the root *R*, thus $P = R$.
2. Page *P* exists, we proceed to the next point.
3. Since $k_1 < x < k_2$, then $P = p_1 = B$.
4. Page *P* exists, we proceed to the next point.
5. Since $x = k_3$ the search ends in success.

Example of searching in a B-tree – search for 57



Procedure

1. We begin the search at the root R, thus $P = R$.
2. Page P exists, we proceed to the next point.
3. Since $x > k_2$, then $P = p_2 = C$.
4. Page P exists, we proceed to the next point.
5. Since $k_1 < x < k_2$, then $P = p_1 = null$.
6. Since P does not exist, the search ends in **failure**.

B-trees – inserting key x

1. First, it is necessary to determine, using the search algorithm, the leaf page L where the key x will be inserted.
2. Two cases can occur:
   - Page L is not completely filled – key x is inserted into the page so that the ordering of keys is preserved.
   - Page L is completely filled, then
     2.1 key x is sorted (e.g., in an auxiliary array) among the keys from page L so that the ordering of keys is preserved. We obtain a sequence of $2n + 1$ keys $k'_1 < k'_2 < \cdots k'_{2n+1}$
     2.2 a new page P is created, with the same parent R as L
     2.3 distribution of keys to pages

Keys	Action
$k'_1, \dots, k'_n$	remain in page L
k'_{n+1}	insert into parent page R
$k'_{n+2}, \dots k'_{2n+1}$	insert into new page P

B-trees – insertion algorithm, notes

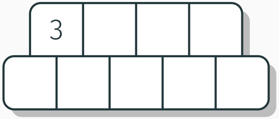
- The process of creating a new page and redistributing keys is called **page splitting**.
- By inserting the key k'_{n+1} into the parent page R, the number of keys in this page increases, which in turn increases the number of references to the child pages of this page. Without moving k'_{n+1}, the page R would lack a free reference for attaching the page P.
- The insertion of the key k'_{n+1} into page R is performed using the same algorithm as the insertion of key x into L. The insertion of k'_{n+1} can cause the page R to split.
- Page splitting can lead to the creation of a new root of the entire tree, which is the only way for a B-tree to increase its height.

Example of Insertion into a B-Tree

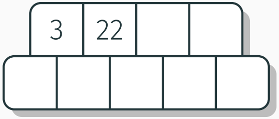
- In this more extensive example, we will gradually build a B-tree with the same parameters as in the search example.
- We will gradually insert the keys 3, 22, 10, 2, 17, 5, 66, 68, 50, 7, 55, 45, 70, 44, 6, 21, 67, 1, 4, 8, 9, 12, and 15 into the tree.

Example of insertion into a B-tree – insertion of keys 3, 22 and 10

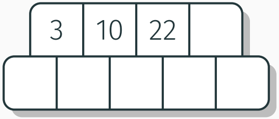
Insertion of key 3



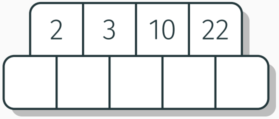
Insertion of key 22



Insertion of key 10

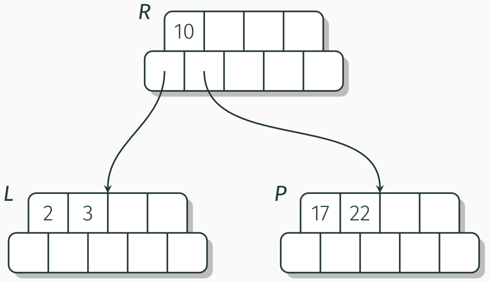


Example of insertion into a B-tree – insertion of key 2



The page is completely full, inserting any additional key will cause a change in the structure of the B-tree.

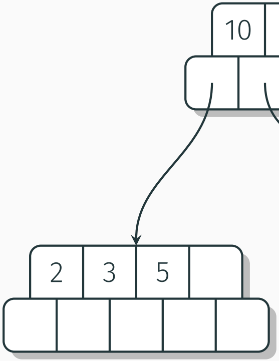
Example of insertion into a B-tree – insertion of key 17



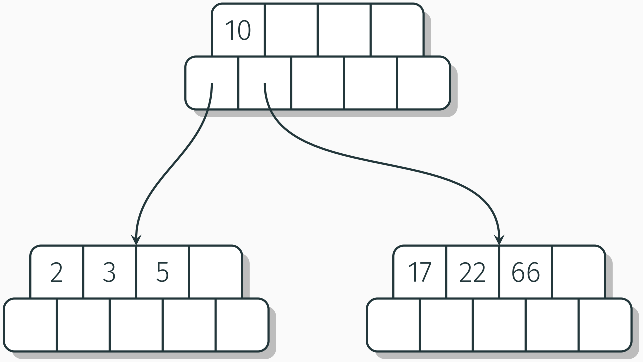
By inserting the key 17, the following occurred:

1. the page ***L*** was split and half of the keys were moved to a new page ***P***,
2. a new root page ***R*** was created and the key 10 was moved to the new root.

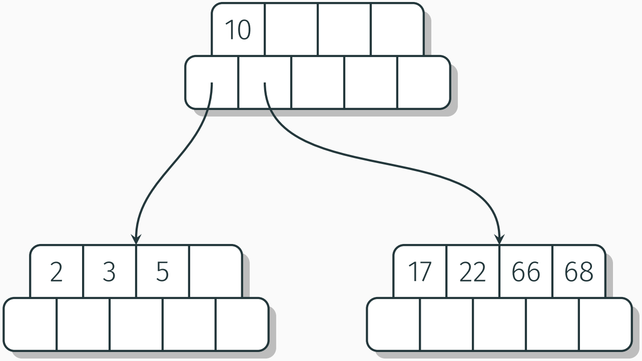
Example of insertion into a B-tree – insertion of key 5



Example of insertion into a B-tree – insertion of key 66

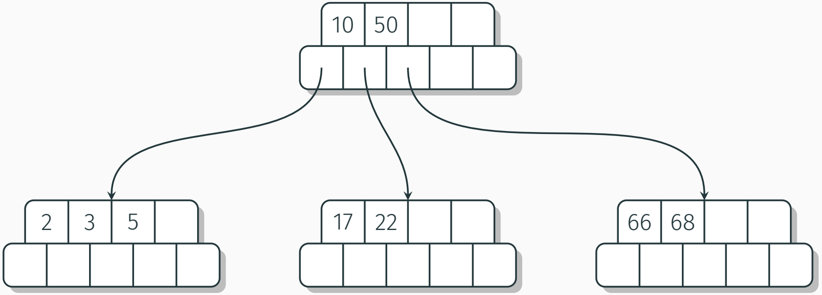


Example of insertion into a B-tree – insertion of key 68



The page with keys 17 to 68 is completely full, inserting another key into this page will cause a change in the structure of the B-tree.

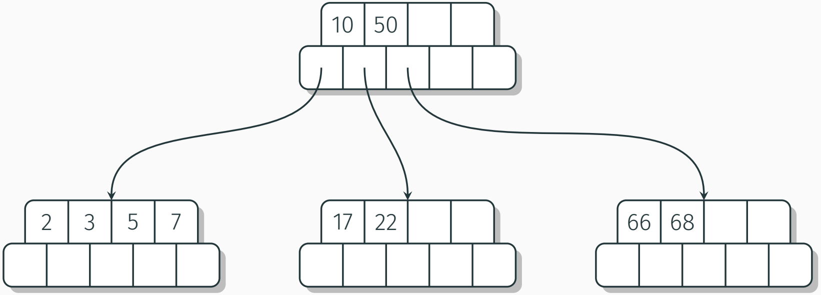
Example of insertion into a B-tree – insertion of key 50



By inserting the key 50, the following occurred:

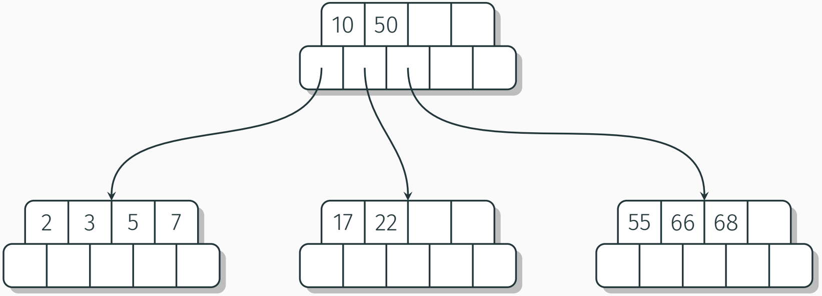
1. the page split and half of the keys were moved to a new page, and
2. at the same time, the newly inserted key 50, being the median of the values in the original page, was moved to the root page.

Example of insertion into a B-tree – inserting key 7

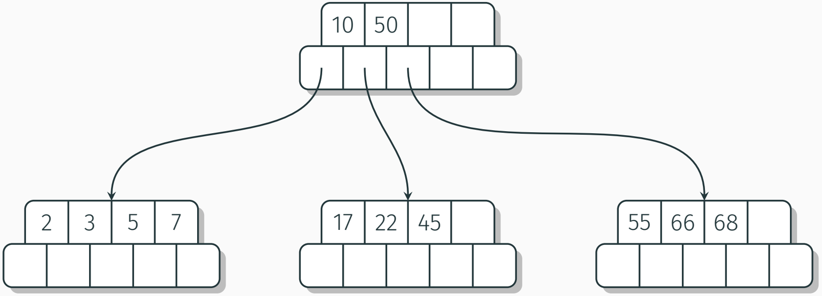


The page with keys 2 to 7 is completely full, inserting another key into this page will cause a change in the structure of the B-tree.

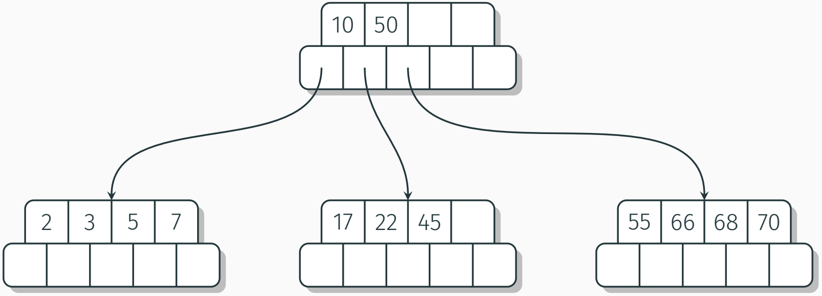
Example of insertion into a B-tree – insertion of key 55



Example of insertion into a B-tree – insertion of key 45

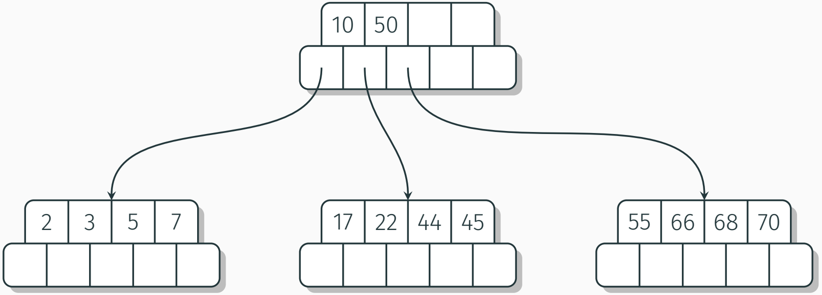


Example of insertion into a B-tree – insertion of key 70



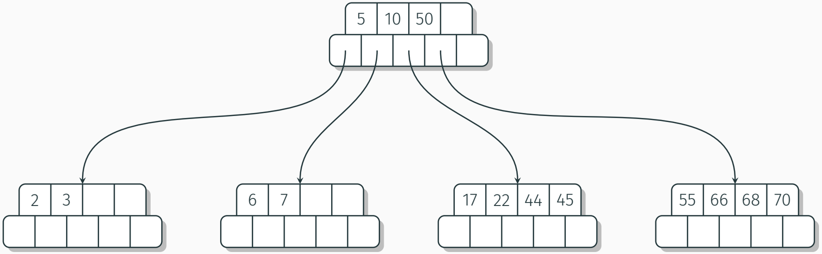
The page with keys 55 to 70 is completely full, inserting another key into this page will cause a change in the structure of the B-tree.

Example of insertion into a B-tree – insertion of key 44



All leaf pages are completely filled, inserting any additional key will cause a change in the structure of the B-tree.

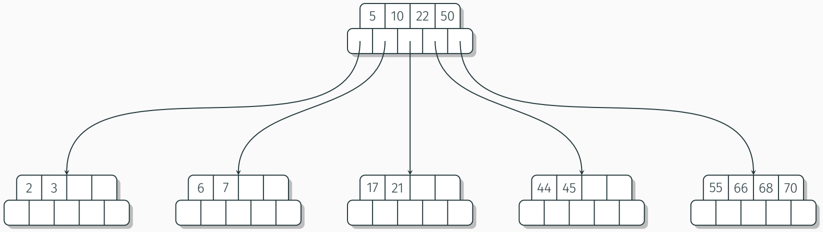
Example of insertion into a B-tree – insertion of key 6



Upon inserting key 6, the following occurred:

1. the page split and half of the keys were moved to a new page, and
2. simultaneously, key 5 was moved to the root page.

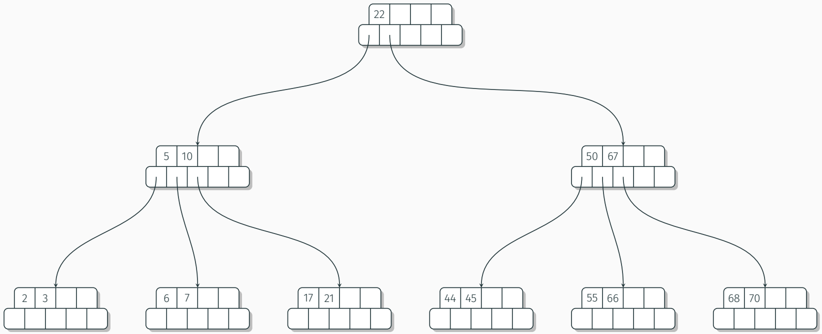
Example of insertion into a B-tree – insertion of key 21



Upon the insertion of key 21, the following occurred:

1. the page split and half of the keys were moved to a new page, and
2. key 22 was moved to the parent page.
3. At the same time, the root page of the tree became full.

Example of insertion into a B-tree – insertion of key 67



Example of insertion into a B-tree – insertion of key 67 (cont.)

Upon inserting key 67, the following occurred:

1. the page split and half of the keys were moved to a new page, and
2. simultaneously, the newly inserted key 67, being the median value in the original page, was moved to the parent page.
3. Since this page was also fully occupied, it split, resulting in the creation of a new root page with a single key 22.

Example of insertion into a B-tree – insertion of key 67 (cont.)

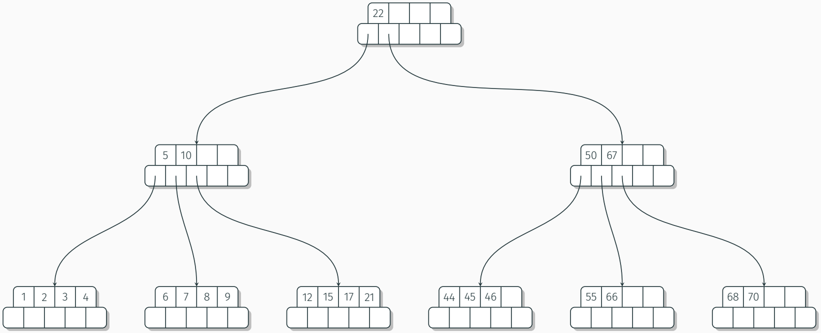
Remarks

- At this point, the B-tree's fill factor reaches its minimum value of approximately 50%. The B-tree has maximum free space for inserting additional keys.
- However, the minimal filling of the B-tree will cause the removal of any key to result in page merging, including the cancellation of the root and a subsequent decrease in the height of the B-tree[1].

[1] See the next part of the presentation

Example of insertion into a B-tree – insertion of additional keys

Into the tree were further inserted keys 1, 4, 8, 9, 12, 15 and 46. The order of key insertion, in this case, does not matter.



B-trees – deletion of key x

1. First, it is necessary to find the key x in the tree.
2. Let us denote the page with key x as P.
3. Two cases can occur:
    - page P is an **internal page** of the tree or
    - page P is a **leaf page**.

B-trees – deletion of key x from internal page P

1. We replace key $\boldsymbol{x}$ in page $\boldsymbol{P}$ with the closest larger key $\boldsymbol{y}$ to it.
2. Key $\boldsymbol{y}$ must be located in the subtree with keys greater than $\boldsymbol{x}$ and, at the same time, is the smallest among these keys, so it must be located in a leaf page.
3. We have thus reduced the deletion of key $\boldsymbol{x}$ from an internal page of the tree to the deletion of key $\boldsymbol{y}$ from a leaf page of the tree.

B-trees – deleting key x from leaf page P

1. We delete key x from page P.
2. If page P still contains at least n keys after deletion, the deletion process is terminated.
3. If P then contains only $n - 1$ keys, we must replenish the missing key.
   - 3.1 We determine the number of keys in the sibling page of P. We denote the sibling as S. The common parent of pages P and S is denoted as R.
   - 3.2 If there are **more than n** keys in S, then
     - 3.2.1 we move the nearest larger key than x from R to P and
     - 3.2.2 we move the smallest key from S to R.
   - 3.3 If there are **exactly n** keys in S, then

B-trees – deleting key x from leaf page P (cont.)

3.3.1 we move keys from page S to page P and obtain one page with $2n - 1$ keys.

3.3.2 We eliminate page S.

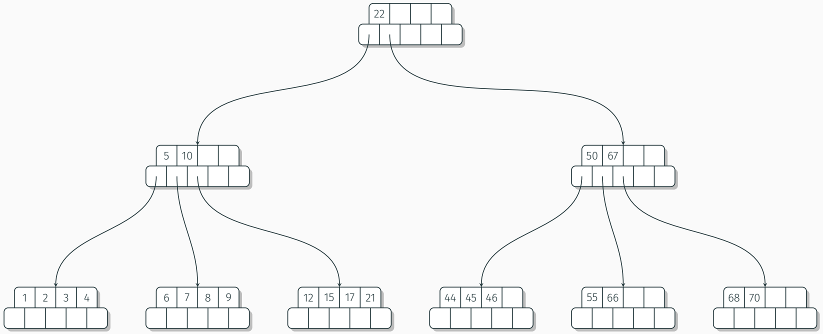
3.3.3 In page R, there is now one redundant pointer to a page. We move the nearest larger key than x from R to page P, which now contains exactly $2n$ keys.

B-trees – deleting key x from leaf page P (cont.)

Remarks

- Usually, we choose a sibling with larger keys than x, i.e., the sibling to the "right" of P. In the previous explanation, we assumed this choice.
- However, it is possible to choose a sibling with smaller keys, i.e., the one to the "left" of P. The further procedure is a mirror image of the "right" sibling.
- The process of moving keys from S to P and subsequent elimination of page S is called **page merging**.
- The process of page merging can continue progressively up to the root of the tree and may lead to the extinction of the current root of the tree. The new root of the tree will then be the page resulting from the merging process. The B-tree thus reduces its height.

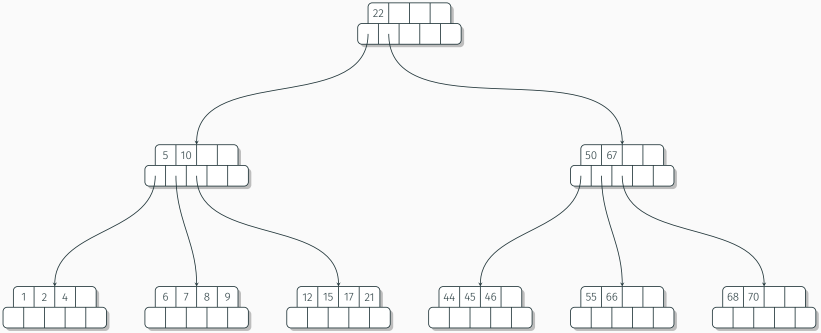
Example of deletion in a B-tree – initial B-tree



In this state, we have left the B-tree at the end of the example of inserting keys into the B-tree. Now we will gradually delete keys from the B-tree.

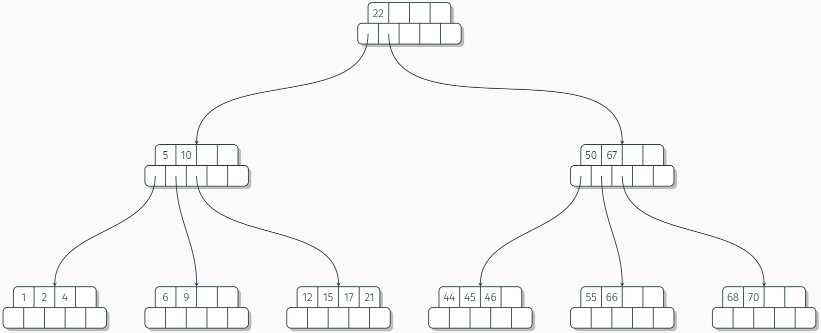
Example of deletion in a B-tree – deletion of key 3

Key 3 is located in a leaf page, where there are enough keys to simply delete key 3. This results in the following B-tree.



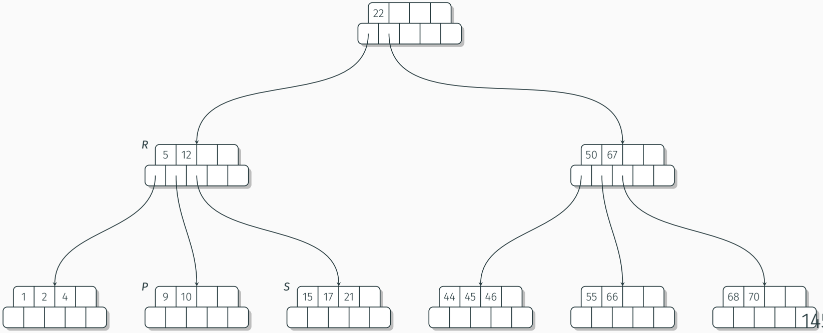
Example of deletion in a B-tree – deletion of keys 7 and 8

In the same way, we delete keys 7, 8 and obtain



Example of deletion in a B-tree – deletion of key 6

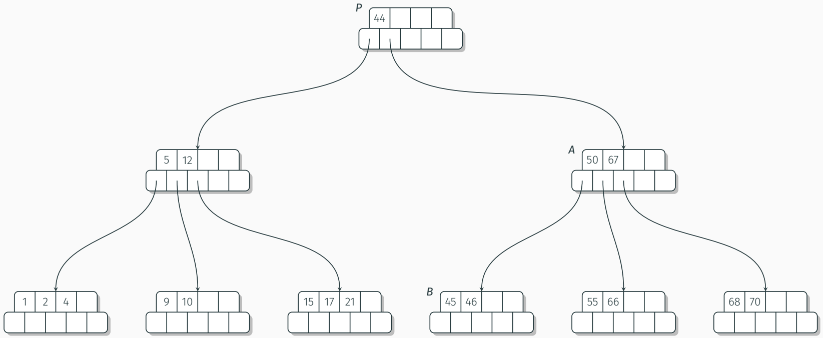
1. After deleting key 6, page *P* contains $n - 1 = 1$ key, number 9.
2. Sibling *S* contains more than n keys.
3. The nearest larger key than 6, i.e. 10, is moved from *R* to *P*.
4. The smallest key from *S*, i.e. 12, is moved to *R*.



Example of deletion in a B-tree – deleting key 22

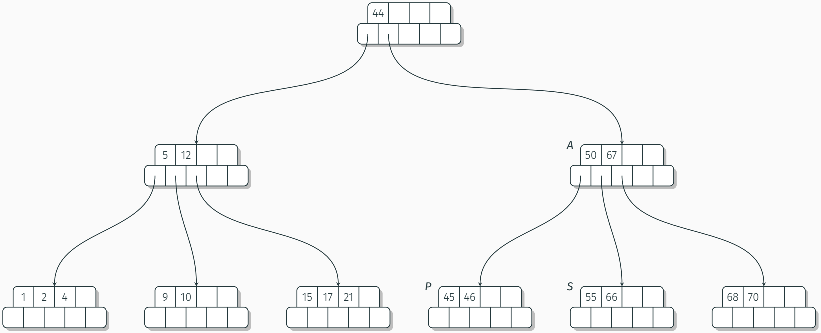
1. Key 22 is located in the internal page ***P***, see the following figure.
2. We replace it with the nearest larger key – larger keys than 22 are in the subtree rooted at page ***A***. From there, we proceed to the leftmost leaf page, in our case to ***B***.
3. We select the smallest key in ***B***, i.e., 44.
4. We have thus reduced the deletion of key 22 to the deletion of key 44.
5. After performing all operations corresponding to the deletion of 44 (in this case, it only involves deleting 44 from page ***B***), we replace key 22 with key 44.

Example of deletion in a B-tree – deleting key 22 (cont.)



Example of deletion in a B-tree – deletion of key 46, phase I

State of the B-tree before deletion begins

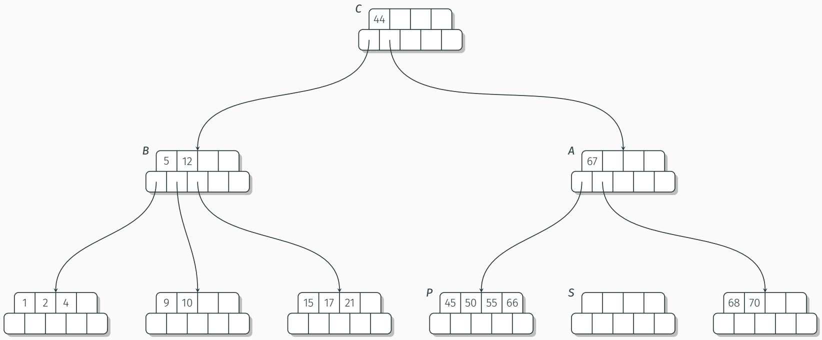


Example of deletion in a B-tree – deletion of key 46, phase I (cont.)

1. After deleting key 46, page *P* contains $n - 1$ keys, i.e., only key 45.
2. Sibling *S* contains exactly less than *n* keys, we must merge pages.
3. We move all keys from *S* to *P*.
4. Page *P* is the first child of page *A*, so we also move the first key from *A* to *P*.

Example of deletion in a B-tree – deletion of key 46, phase I (cont.)

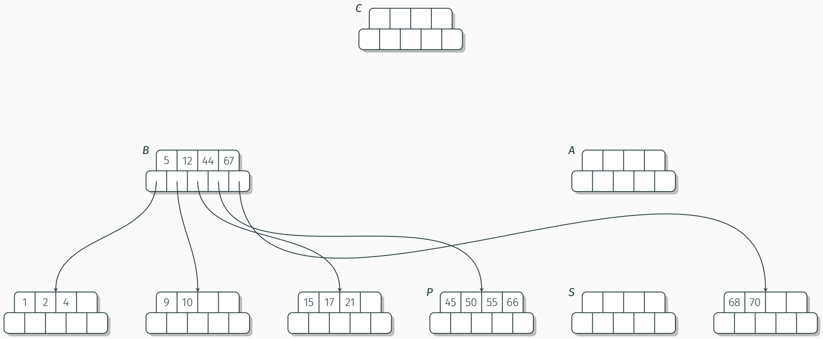
Result of phase 1



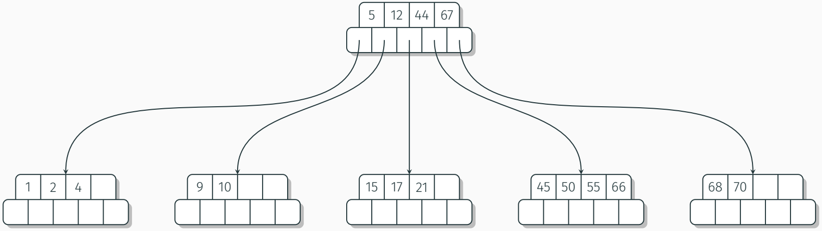
Example of deletion in a B-tree – deletion of key 46, phase II

1. In page *A*, only $n - 1$ keys remain, which contradicts the definition of a B-tree.
2. Sibling *B* contains exactly n keys, so we must also perform page merging at this level.
3. We move key 67 from page *A* to page *B*.
4. And similarly, we also move one key from the parent page *C* to *B*.
5. This results in the elimination of the root page and a decrease in the height of the B-tree.

Example of deletion in a B-tree – deletion of key 46, phase II (cont.)

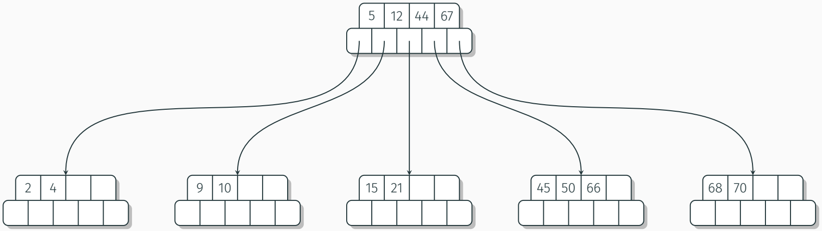


Example of deletion in a B-tree – deletion of key 46, result



Example of deletion in a B-tree – deletion of keys 1, 17, and 55

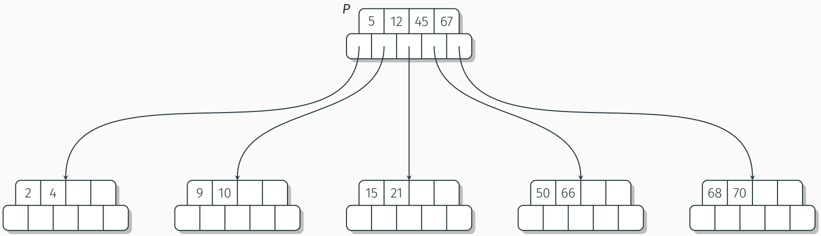
.Keys 1, 17, and 55 are located in leaf pages, where there is a sufficient number of keys to simply delete them.



Example of deletion in a B-tree – deletion of key 44

1. Key 44 is located on an internal page.
2. We replace it with the nearest larger key, i.e. key 45.

Resulting tree

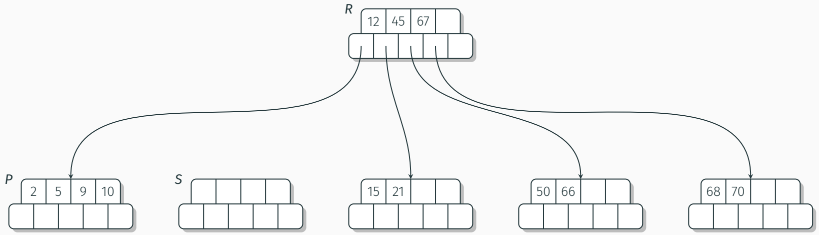


The B-tree is now in a state where the leaf pages are filled to the minimum acceptable level. Deletion of any key will cause page merging.

Example of deletion in a B-tree – deletion of key 4

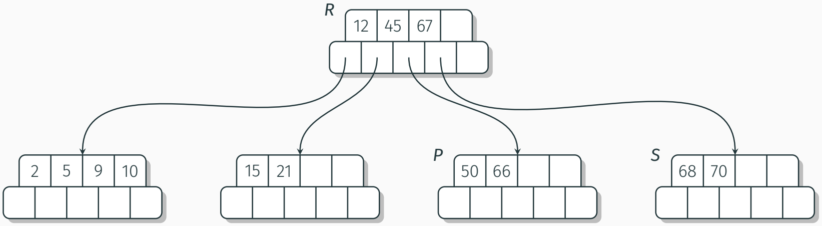
1. After deleting key 4, page ***P*** contains only ***n*** **– 1** keys.
2. Sibling ***S*** contains exactly ***n*** keys.
3. We move the keys from ***S*** to ***P***.
4. We also move key 5 from the parent page ***R*** to ***P***, because otherwise one child pointer in ***R*** would be redundant.

Resulting tree



Example of deletion in a B-tree – deletion of key 45

The tree before deleting key 45



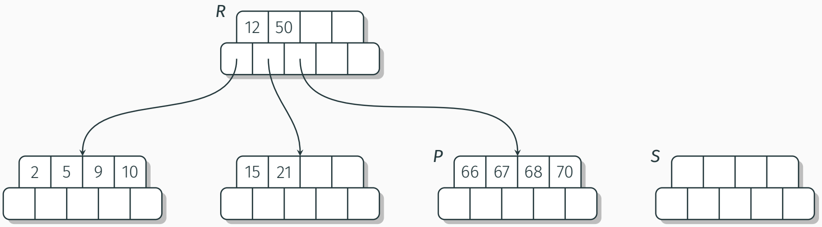
1. Key 45 is located in the internal page *R*.
2. We replace it with the next larger key, i.e., key 50.

Example of deletion in a B-tree – deletion of key 45 (cont.)

3. This reduces the deletion of key 45 to the deletion of key 50.
4. After deleting key 50, page *P* contains only $n - 1$ keys.
5. Sibling *S* contains exactly *n* keys.
6. We move keys from *S* to *P*.
7. We also move the next larger key than 45, i.e., key 57, from parent page *R* to *P*, because otherwise one child pointer in *R* would be left over.

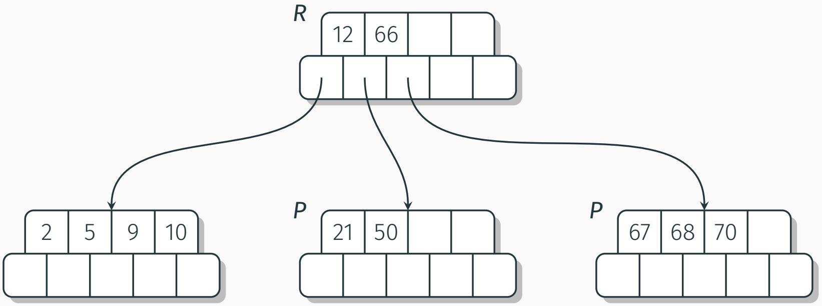
Example of deletion in a B-tree – deletion of key 45 (cont.)

Resulting tree



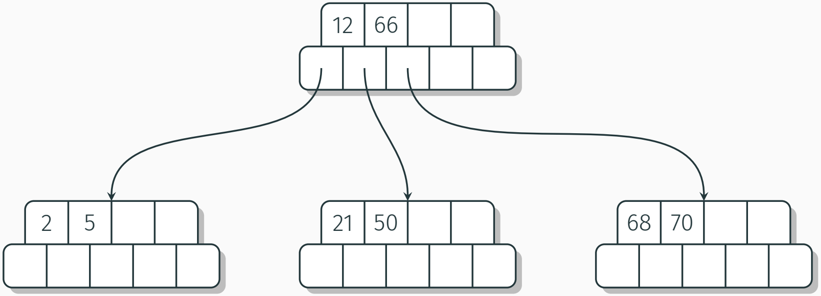
Example of deletion in a B-tree – deletion of key 15

After deleting 15, there remained only $n - 1$ keys in page P. We must therefore move one key from page S through page R.



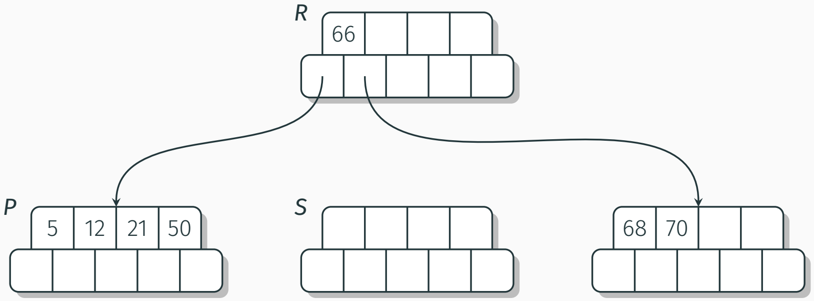
Example of deletion in a B-tree – deletion of keys 9, 10 and 67

Deletion of keys 9, 10 and 67 is very simple.

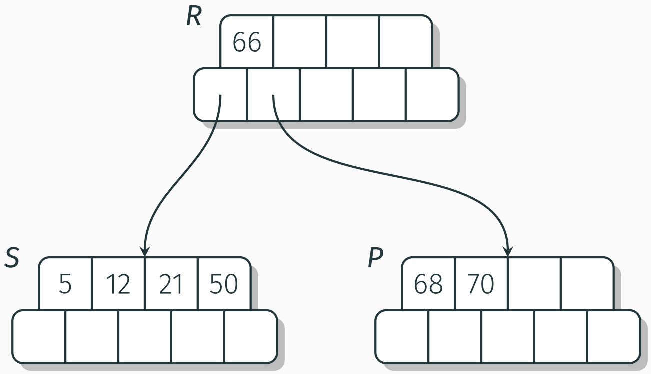


Example of deletion in a B-tree – deletion of key 2

After deleting 2, there are only $n - 1$ keys left in page P. The sibling S contains n keys, so page merging occurs.

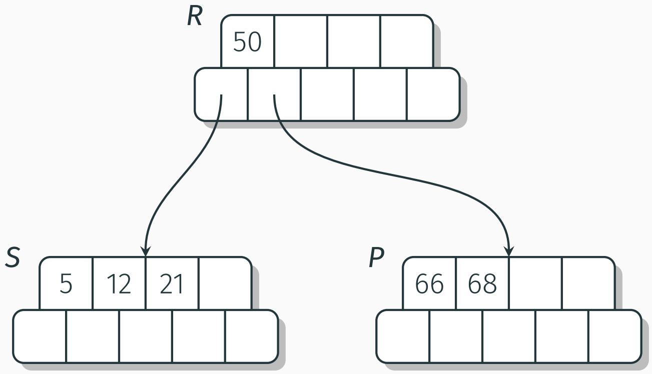


Example of deletion in a B-tree – deletion of key 70



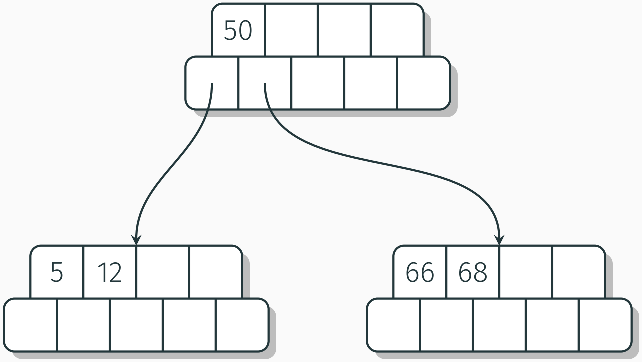
Example of deletion in a B-tree – deletion of key 70 (cont.)

After deleting 70, page *P* is left with only $n - 1$ keys. Sibling *S* contains more than n keys, so a shift of 66 from *R* to *P* occurs and the nearest smaller key from *S* to *R*.



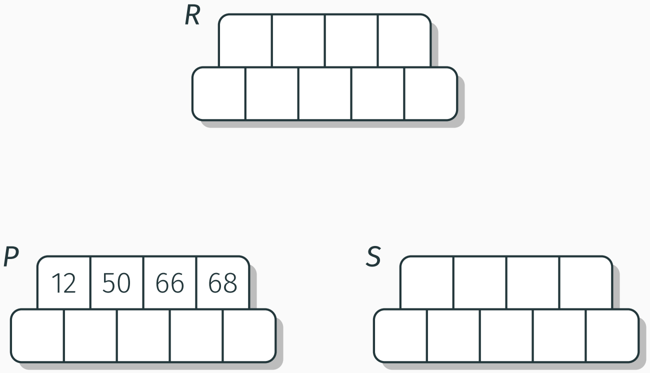
Example of deletion in a B-tree – deletion of key 21

Deletion of key 21 is very simple.



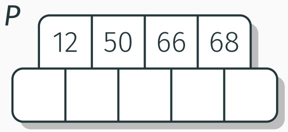
Example of deletion in a B-tree – deletion of key 5

Deletion of key 5 is evident.



Example of deletion in a B-tree – deletion of key 5 (cont.)

Page *P* has become the new root, and simultaneously the only page, of the B-tree.



Deletion of keys 12, 50, 66, and 68 is now a trivial matter.

Thanks for your attention