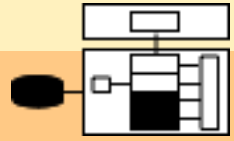
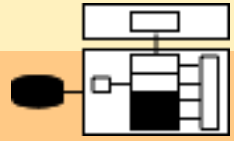
TREE-STRUCTURE ORGANIZATIONS

Khalid Belhajjame



What for?

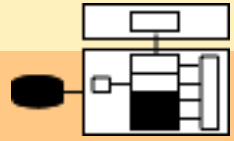
- The techniques we have seen thus far are ineffective for key-range searches.
- For this reason, solutions that are based on tree structures have been explored.
- We will examine two kinds of trees that are widely used in database systems: B Tree and B+ Tree structures



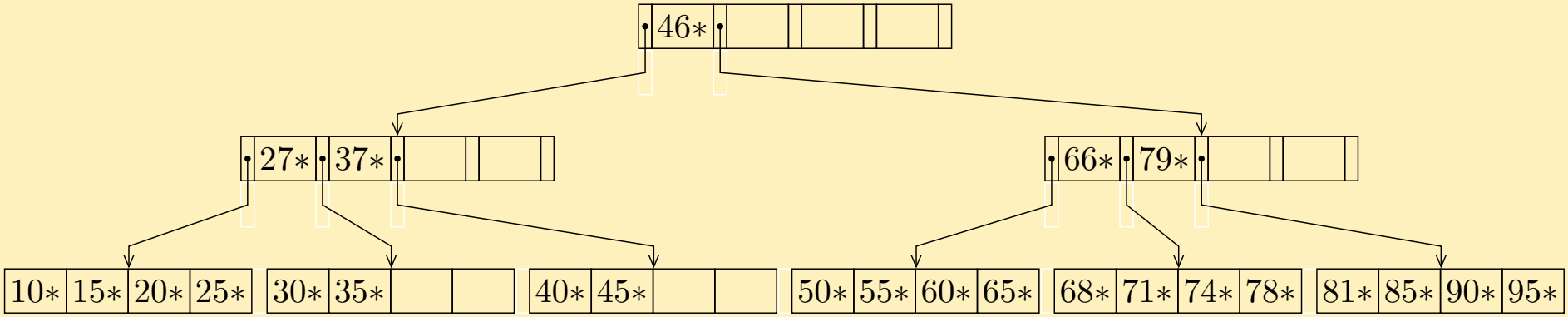
B Trees

A B-tree of *order* m ($m \geq 3$) is a search tree that satisfies the following properties:

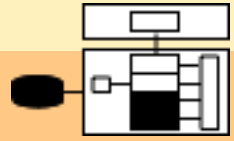
1. Each node contains at most $m - 1$ keys.
2. Each node, except the root, contains at least $\lceil m/2 \rceil - 1$ keys. The root may contain any number n of keys with $n \leq m - 1$.
3. A node is either a leaf node or has $j + 1$ children, where j is the number of keys of the node.
4. All leaves appear on same level.



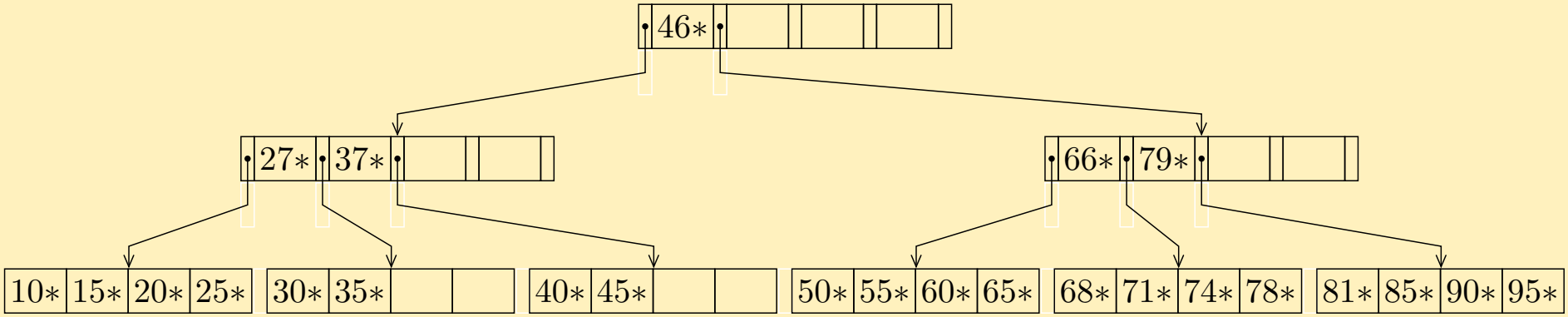
B Trees: Example of a B tree with m = ?



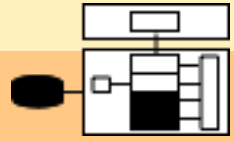
K* denotes the record identified by the key k. For example, 46* is the record identified by 46.



B Trees: Example of a B tree with m = 5



K* denotes the record identified by the key k. For example, 46* is the record identified by 46.

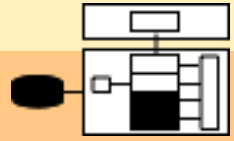


5. Each node has the following structure:

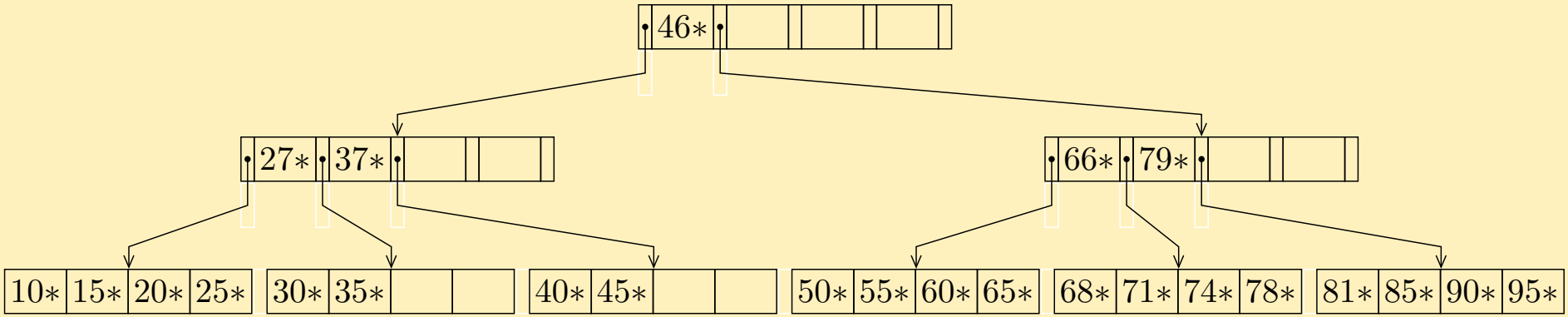
$$[p_0, k_1*, p_1, k_2*, p_2, \ldots k_j*, p_j]$$

where:

- The keys are sorted: $k_1 < \ldots < k_j$.
- p_i is a pointer to another node of the tree structure, and is undefined in the leaves.
- Let $K(p_i)$ be the set of keys stored in the subtree pointed by p_i. For each non-leaf node, the following properties hold:
  - $\forall y \in K(p_0), y < k_1$
  - $\forall y \in K(p_i), k_i < y < k_{i+1}, i = 1, \ldots j-1$
  - $\forall y \in K(p_j), y > k_j$

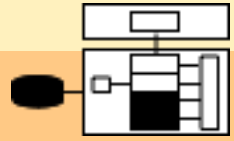


B Trees: Example of a B tree with m = 5

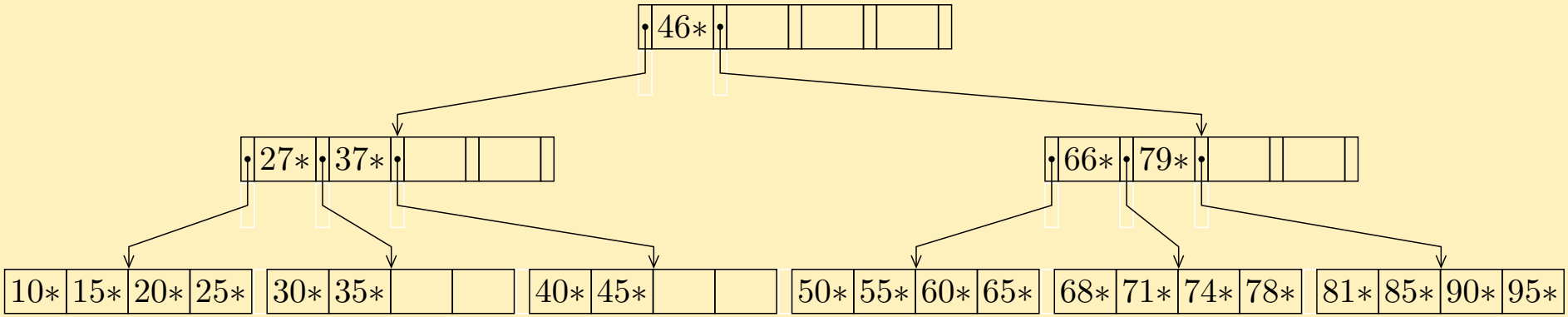


K* denotes the record identified by the key k. For example, 46* is the record identified by 46.

The *height* h of a B–tree is the number of nodes in a path from the root to a leaf node. The height of the above B tree is ?

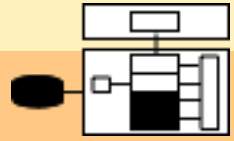


B Trees: Example of a B tree with m = 5



K* denotes the record identified by the key k. For example, 46* is the record identified by 46.

The *height* h of a B–tree is the number of nodes in a path from the root to a leaf node. The height of the above B tree is 3



Operations on the Tree: Search

The search of a key k starts at the root node. If the key is not in the root, and h > 1, the search continues as follows:

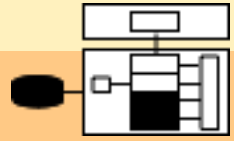
1. If $k_i < k < k_{i+1}$, $1 \le i \le m$, then the search continues in the subtree p_i.

2. If $k_m < k$, then the search continues in the subtree p_m.

3. If $k < k_1$, then the search continues in the subtree p_0.

If the key value is not found in a leaf node, the search is unsuccessful, otherwise the search cost is ≤ h.

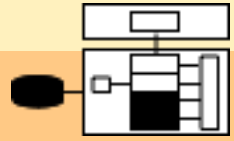
搜索成本小于等于h。



Operations on the Tree: Insertion

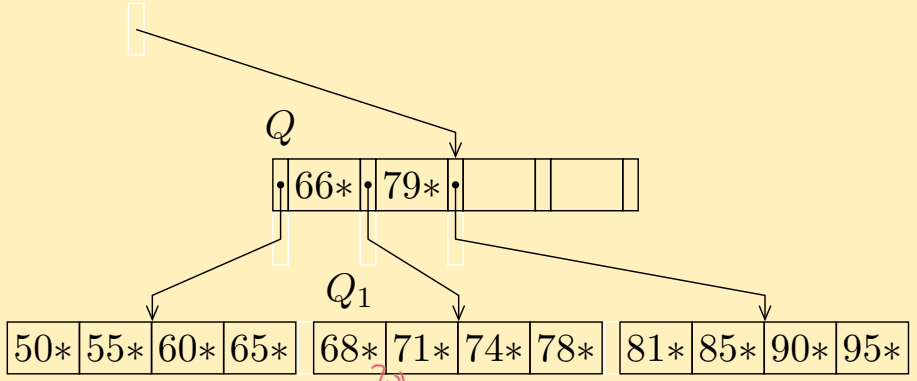
The insertion of a key k into a B-tree is performed as follows

- First, a search is made for the leaf node which should contain the key k.
  - An unsuccessful search determines the leaf node Q1 where k should be inserted.
- If the node Q1 contains less than m – 1 keys, then k is inserted and the operation terminates.
- Otherwise, if Q1 is full, it will be split into two nodes, with the first half of the m keys that remain in the old node Q1, the second half of the keys that go into a new adjacent node Q2, and the median key, together with the pointer to Q2, that is inserted into the father node Q of Q1, repeating the insertion operation in this node.
- This splitting and moving up process may continue if necessary up to the root, and if this must be split, a new root node will be created and this increases the height of the B-tree by one.

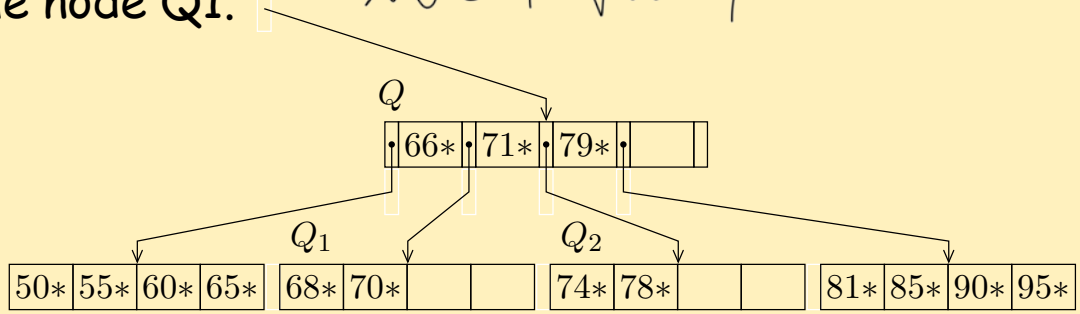


Example of Insertion

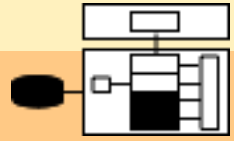
What is the effect of the insertion of the key 70 in the following B-tree



The key 70 must be inserted in the node Q1.



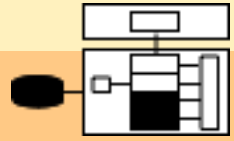
- The node Q1 is full, and it is split into two nodes Q1 and Q2, and the median key 71, together with the pointer to Q2, is inserted into the node Q.



Operations on the Tree: deletion

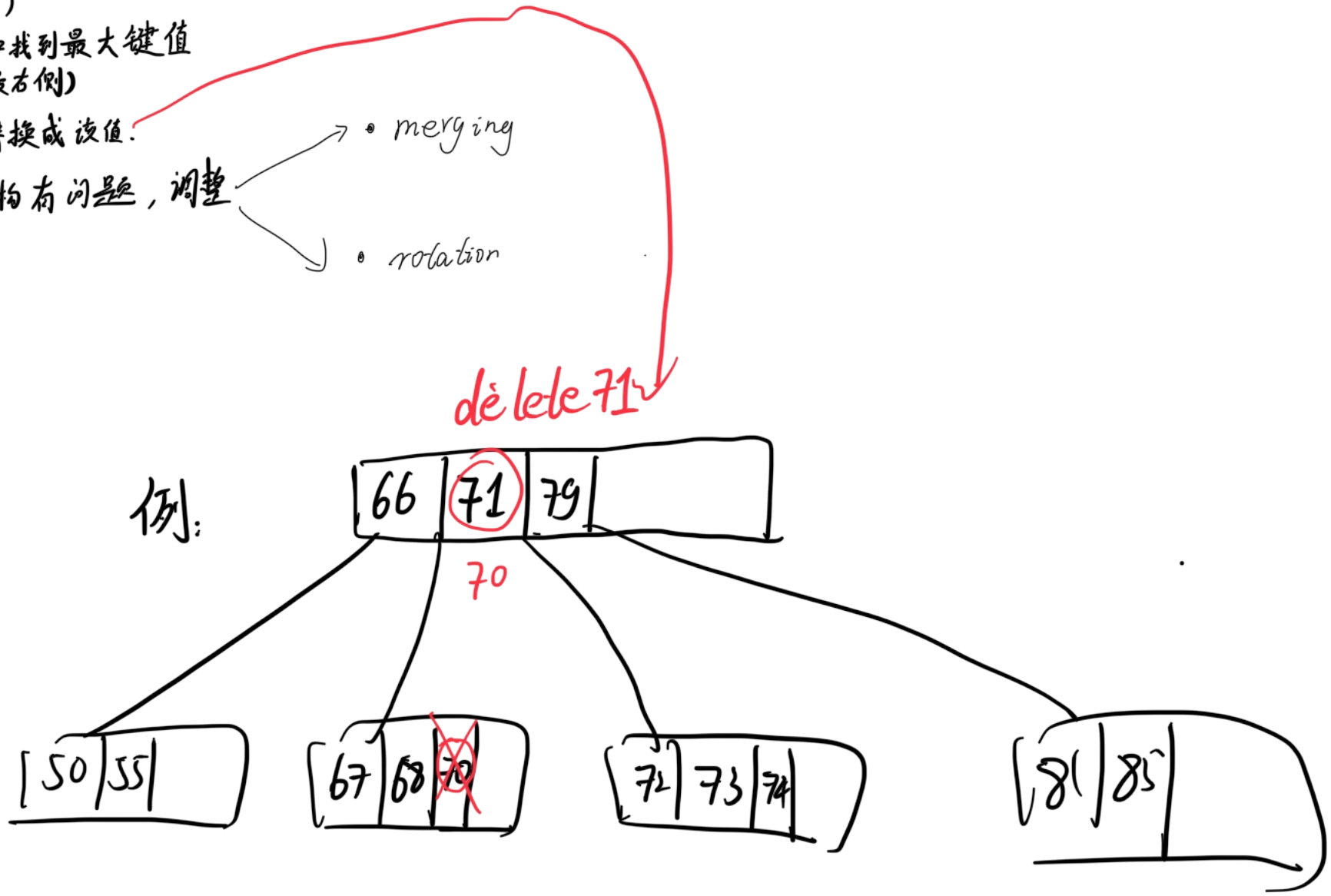
Key deletion is slightly more complicated.

- If the key to be deleted does not occur in a leaf, we replace it with the largest value in its left subtree and then proceed to delete that value from the node that originally contained it.
  - In a B-tree, the largest value in any value's left subtree is guaranteed to be in leaf. Therefore wherever the value to be deleted initially resides, the following deletion algorithm always begins at a leaf.
- Furthermore, after a deletion if the leaf node p has less than $\lceil m/2 \rceil - 1$ keys, it has to be regrouped with an adjacent brother node in order to respect the definition of B-tree
- We can do so using one of the following techniques: merging or rotation.



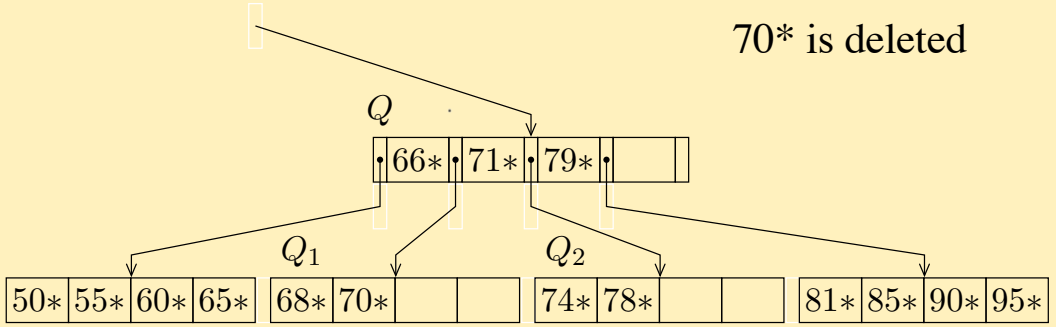
删除操作：
(键不存在叶子节点中)

- 删除键的左子树中找到最大键值
  (左子树叶子节点的最右侧)
- 把要删除的键替换成该值.
- 如果造成了树结构有问题，调整
  - merging
  - rotation

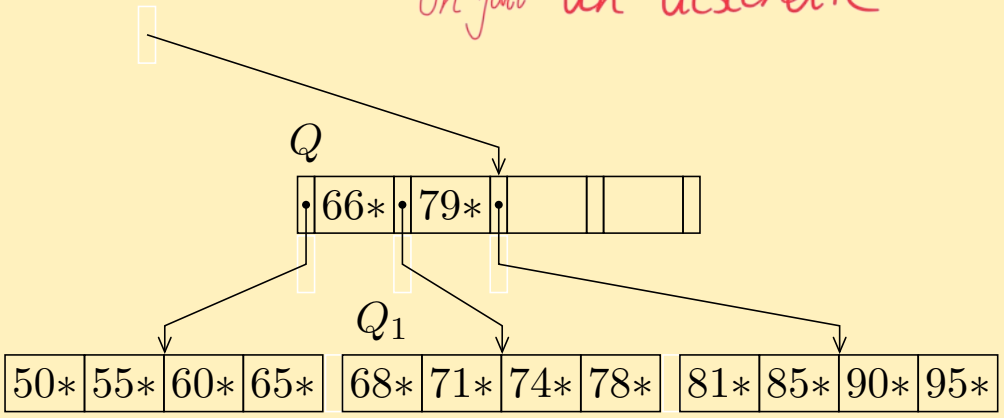


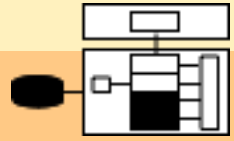
Node Merging

- Merging is illustrated on the right.
- If key 70 is deleted from node Q1, it becomes underfull and it is merged with the brother to the right Q2.
- The key 71 separating the two nodes in the ancestor Q is no longer necessary and it too is added to the single remaining leaf Q1, so the tree will become that shown in



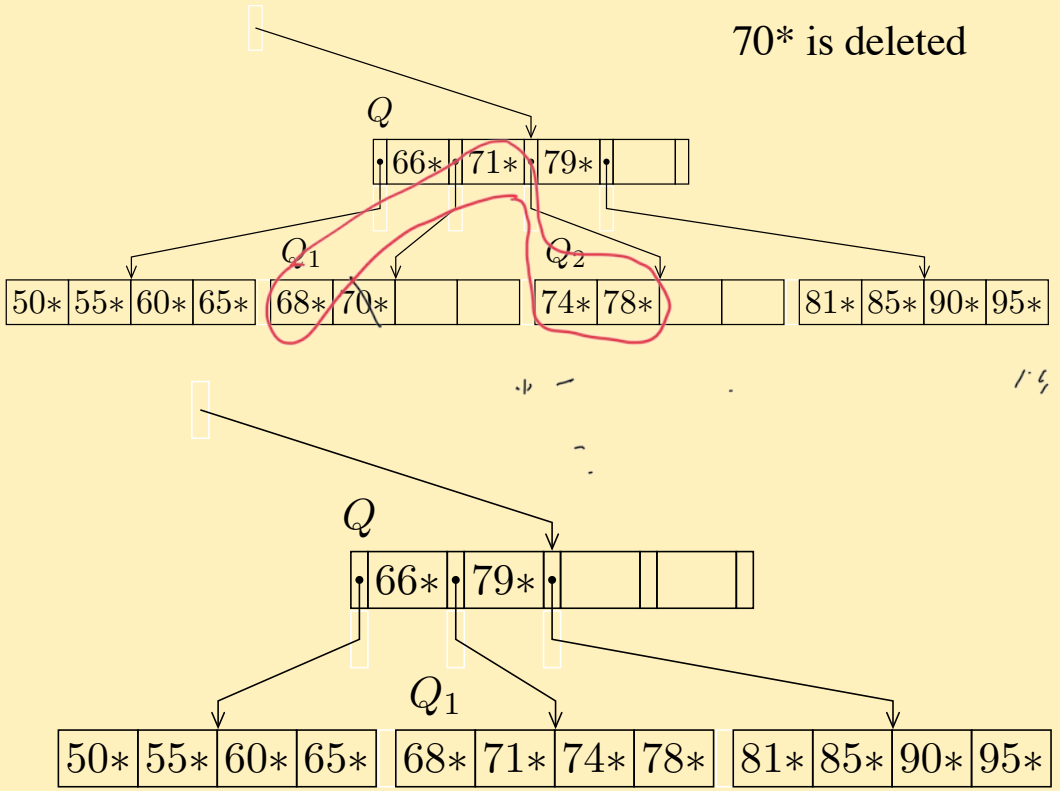
on fait un descendre



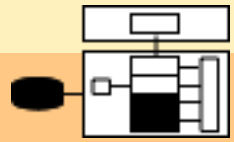


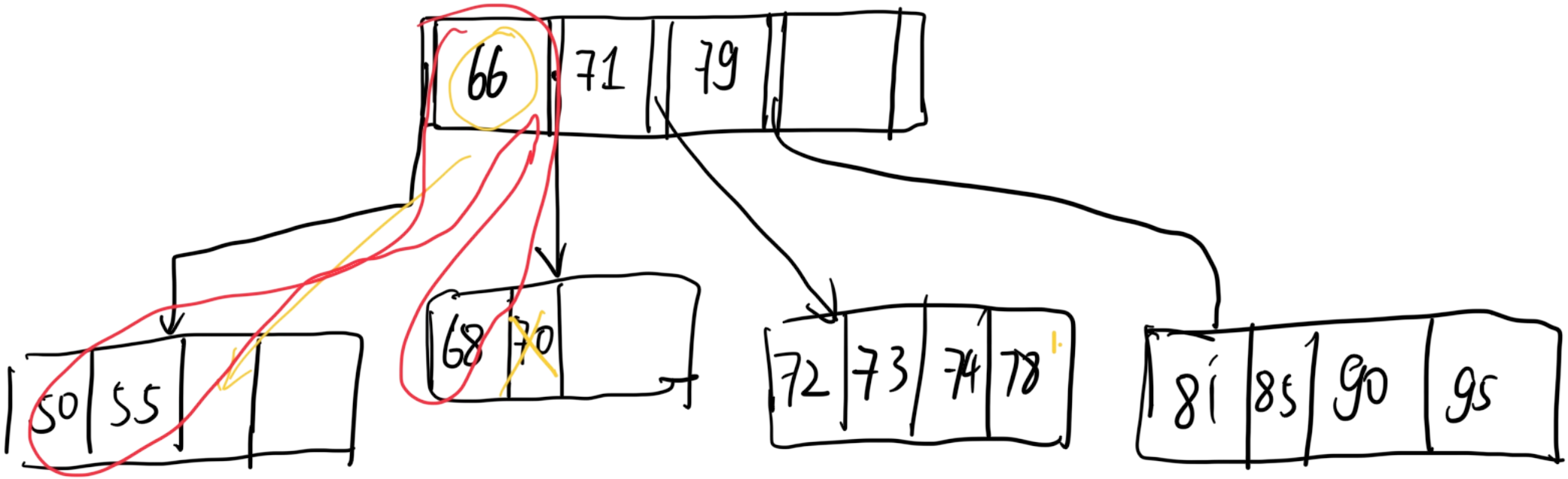
Node Merging Technique

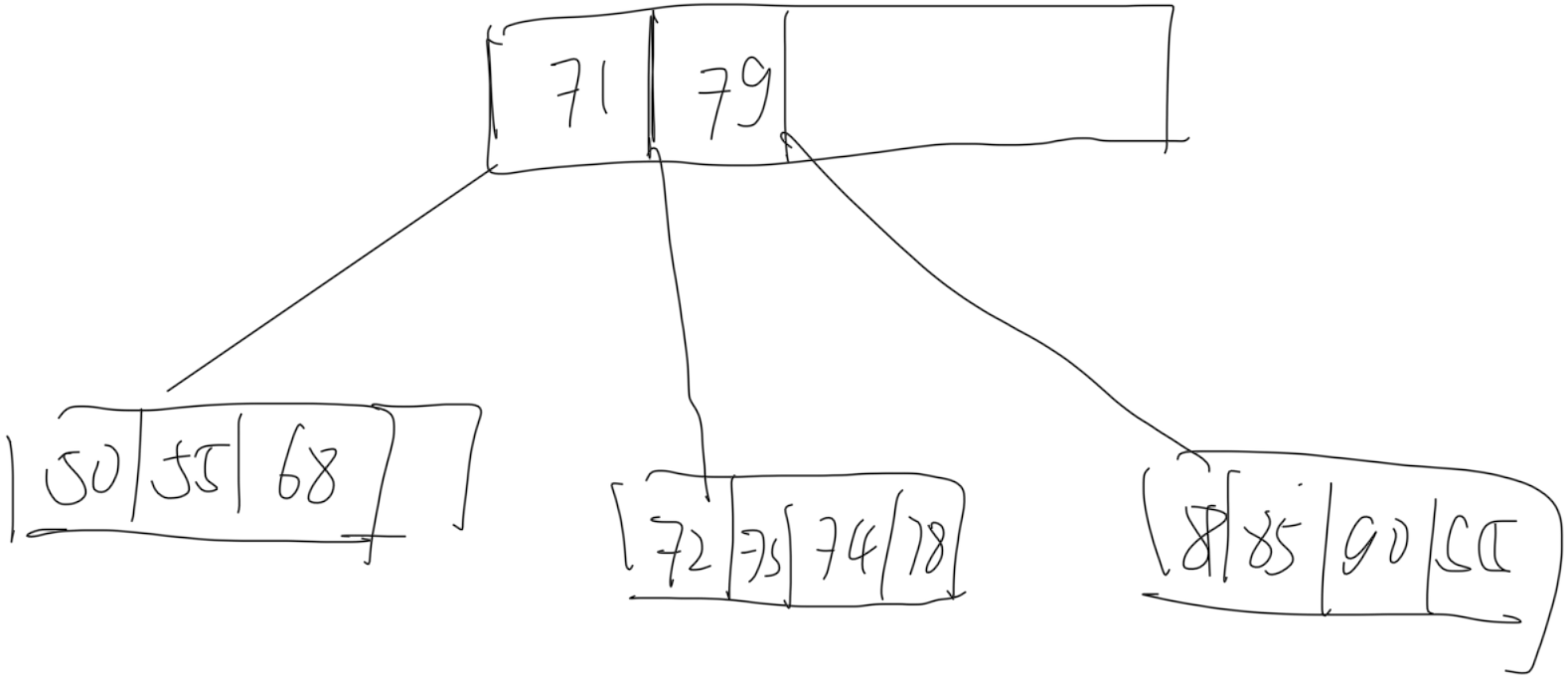
- The elimination of 71 from Q can cause a further underflow by requiring the merging of Q with one of its adjacent brothers.
- In such a case, the process is applied recursively and terminates upon encountering a node that does not need be merged or if the root node is used.
- If the root node contains a single key, as a result of the merging, it becomes empty, and is removed.
- The result is that the B-tree shrinks from the top. Thus the deletion process reverses the effects of the insertion process.



Is node merging always possible?

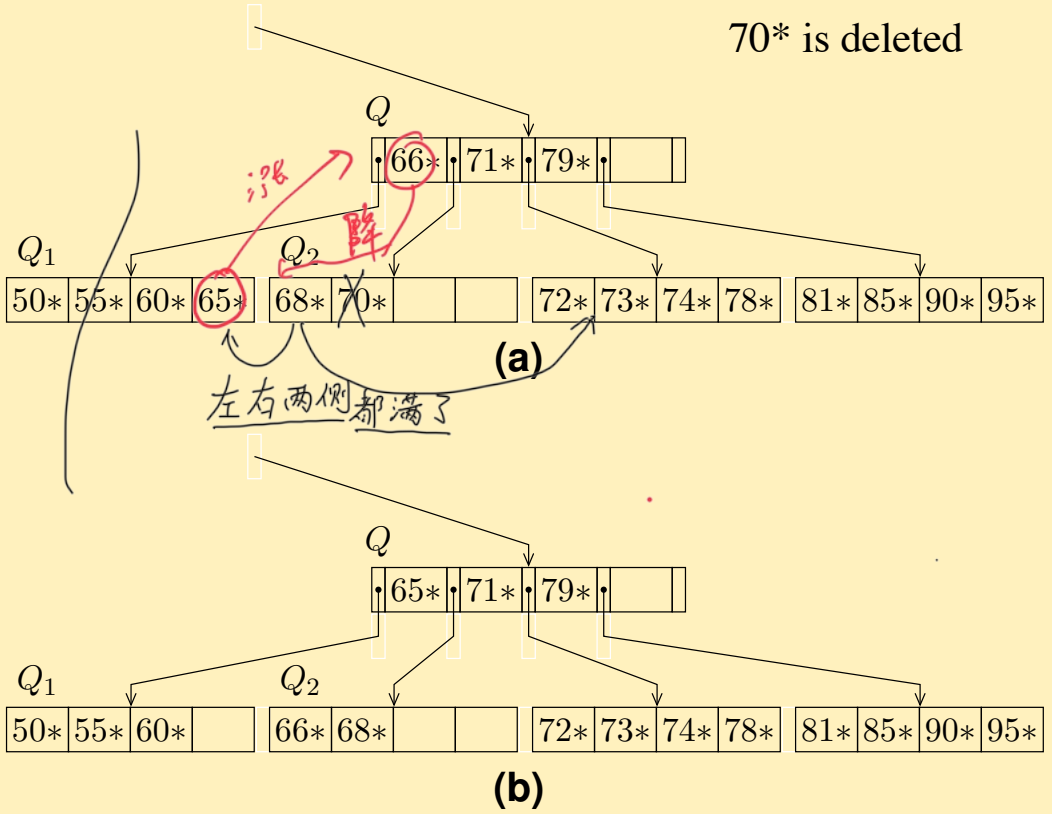



66
71
79
50
55
68
70
72
73
74
78
81
85
90
95


71
79
50
55
68
72
73
74
78
81
85
90
95

Rotation technique

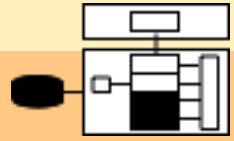
- When the merging of the node p with one of its adjacent brothers is not possible, then the rotation technique is applied.
- Rotation is illustrated on the right.
- If key 70 is deleted from Q2, it becomes underfull and a rotation is performed to borrow the maximum key 65 from the brother to the left Q1.
- The key 65 is moved in the ancestor node Q and replace the key 66 which is moved in Q2 as the new smallest key value.
- The tree will become that shown in Figure b.



B Tree: Performance

- Equality Search:

  The cost of a search is ?

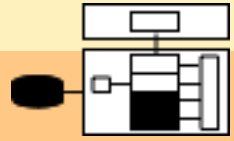


B Tree: Performance

- Equality Search:

  The cost of a search is $1 \leq Cs \leq h$ reads.

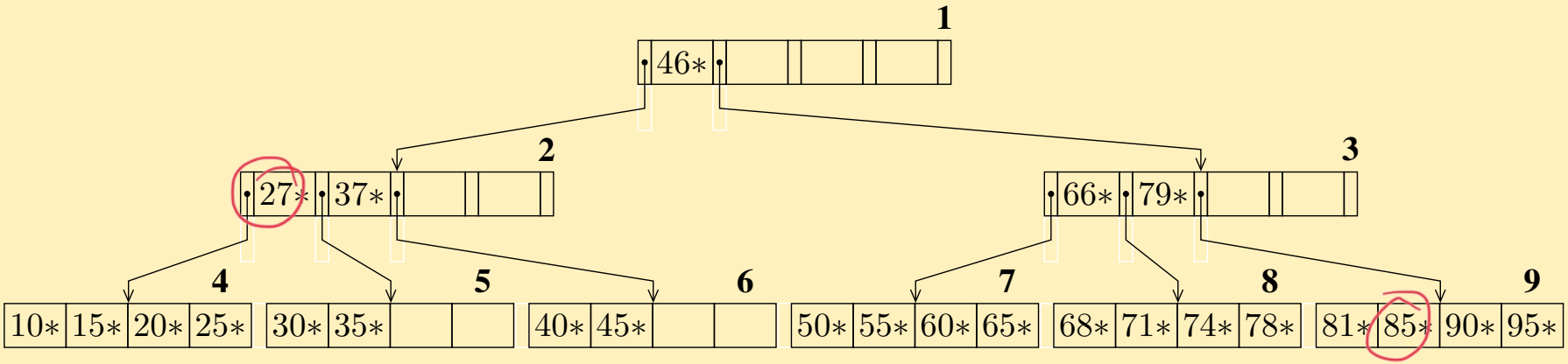
  Cost 在 1 到 h 之间。



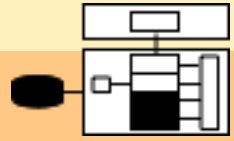
B Tree: Performance

- Range Search: ?

Let us consider the following B Tree



How can we go about searching the records the keys of which belong to the following interval [27,85] using the B-Tree?

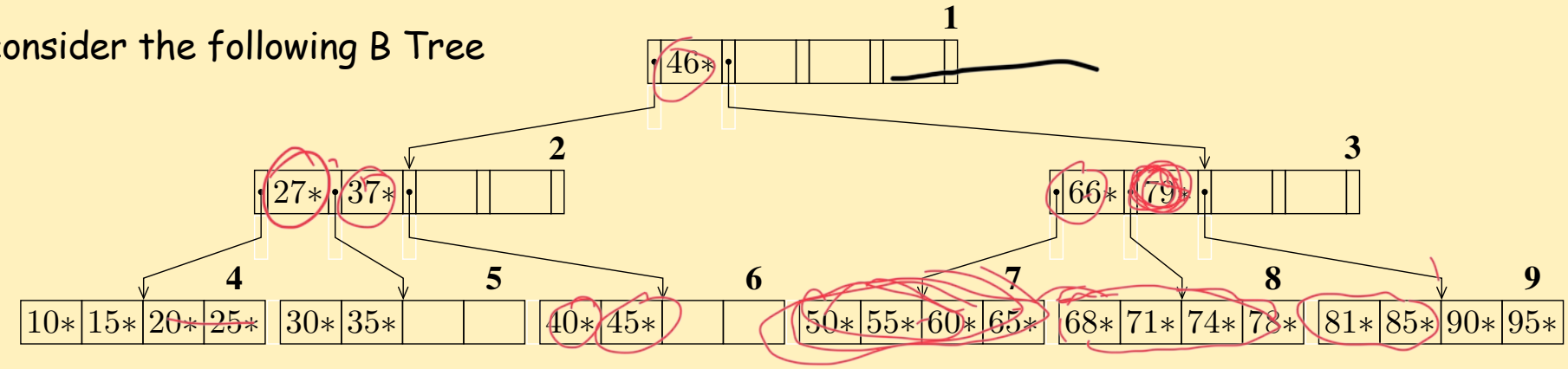


B Tree: Performance

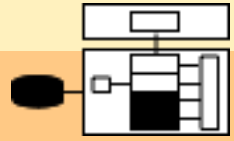
- Range Search: ?

B-trees are very good for equality searches, but not to retrieve data sequentially or for range searches, because they require multiple tree traversals. This is the reason why a variation of this tree structure is usually used.

Let us consider the following B Tree

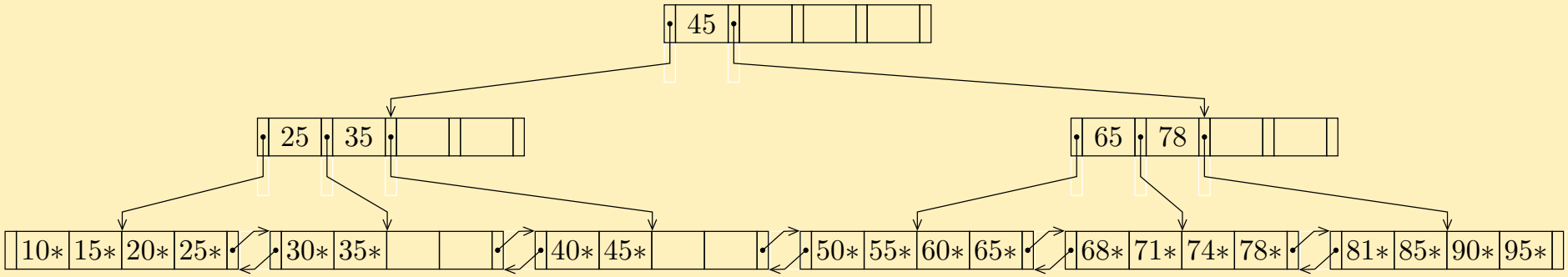


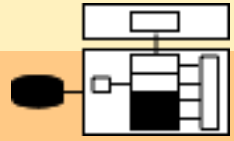
To retrieve all records with the keys in increasing order, the tree is visited in the *in-order* (*symmetric*) traversal and so the nodes are visited as follows: 1, 2, 4, to find the deepest node to the left with the minimum key, and then upward and downward for the sequential search, 2, 5, 2, 6, 1, 3, 7, 3, 8, 3, 9, 1.



B+ Tree

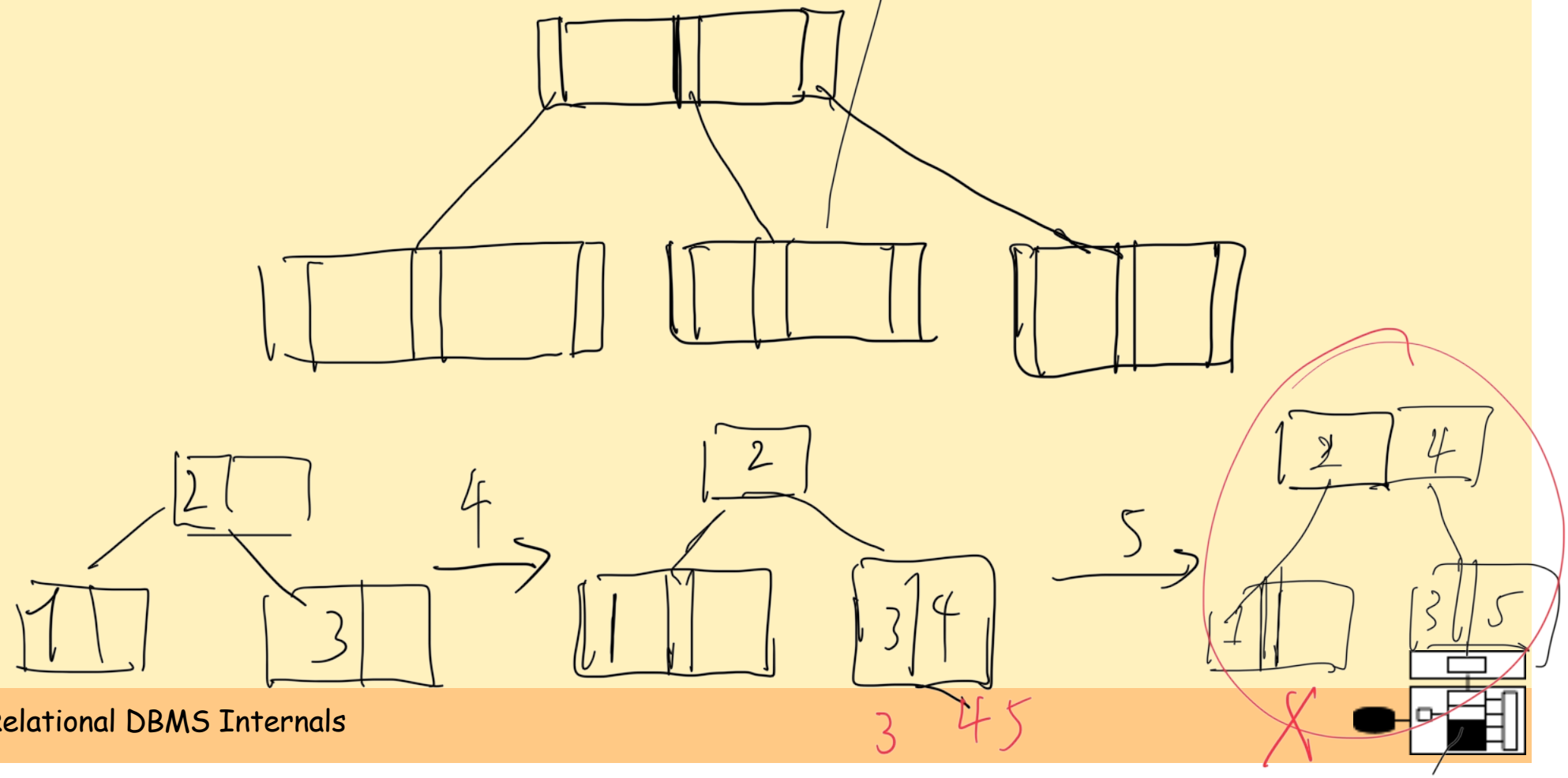
- A B+-tree is a well known B-tree variant to enhance search performance especially for range queries.
- In a B+-tree *all the records*, denoted as k∗, are stored in the leaf nodes, organized into a doubly linked list.
- Only the set of the *highest keys*, denoted as k, in each leaf node, is stored in the non-leaf nodes, organized as a B-tree

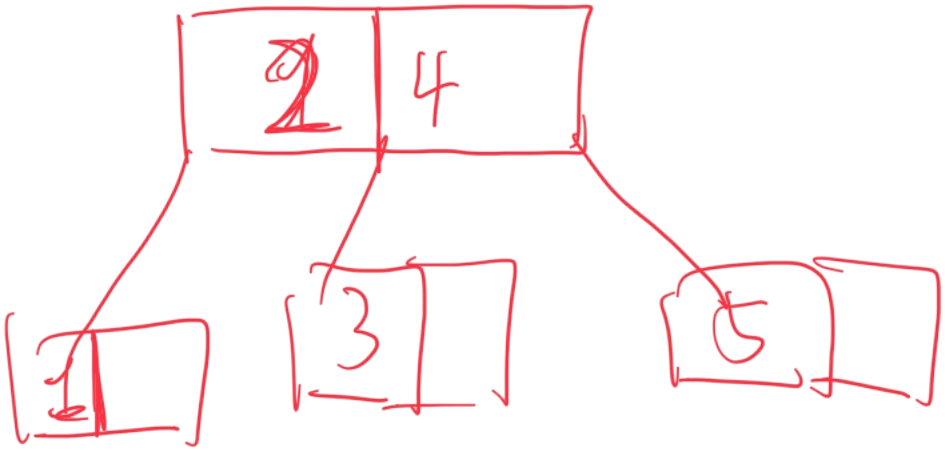


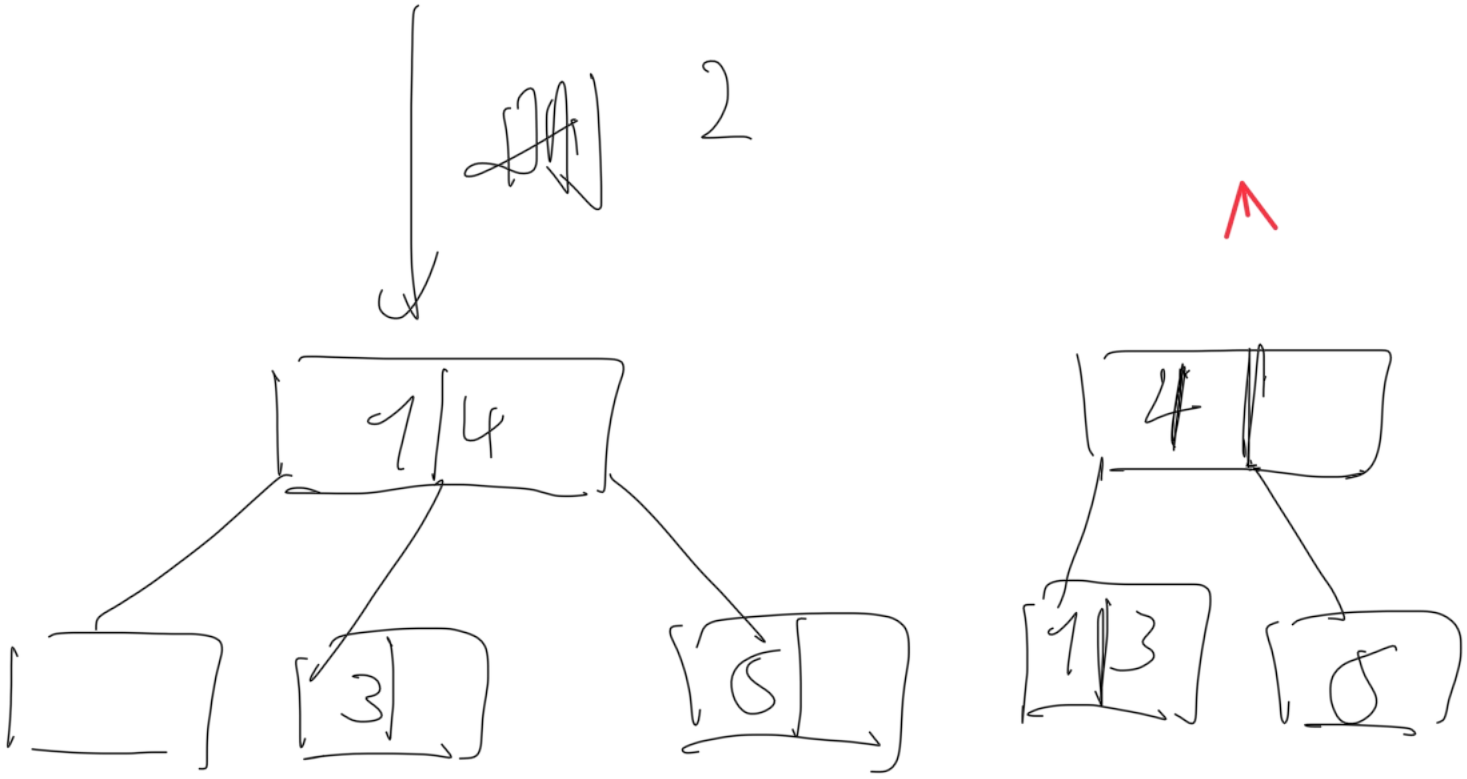


Exercises

- Show the results of entering the records with the keys in the order (1,2,3,4,5) to an initially empty B tree of order m=3.

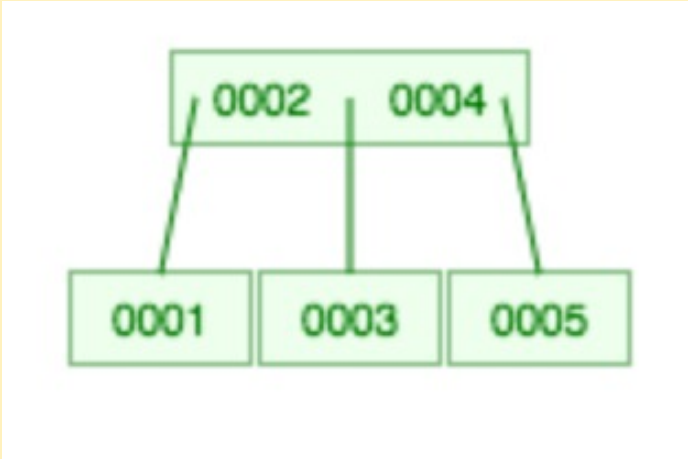



2
4
1
3
5


2
1
4
3
5
4
1
3
5

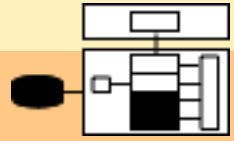
Exercises

- Show the results of entering the records with the keys in the order (1,2,3,4,5) to an initially empty B tree of order m = 3.



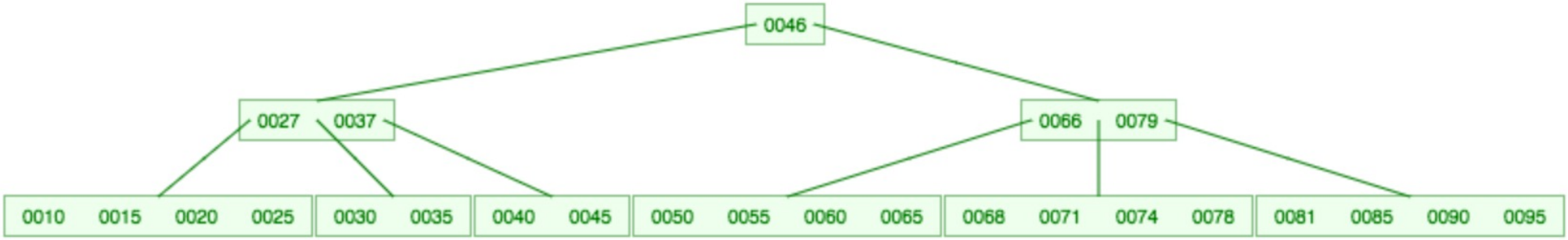
Exercises

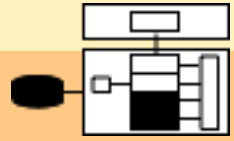
- Show the results of entering the records with the keys in the order (10, 15, 30, 27, 35, 40, 45, 37, 20, 50, 55, 46, 71, 66, 74, 85, 90, 79, 78, 95, 25, 81, 68, 60, 65) to an initially empty B tree of order m = 5.



Exercises

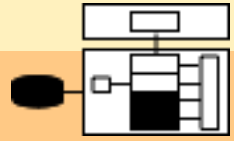
- Show the results of entering the records with the keys in the order (10, 15, 30, 27, 35, 40, 45, 37, 20, 50, 55, 46, 71, 66, 74, 85, 90, 79, 78, 95, 25, 81, 68, 60, 65) to an initially empty B tree of order m = 5.





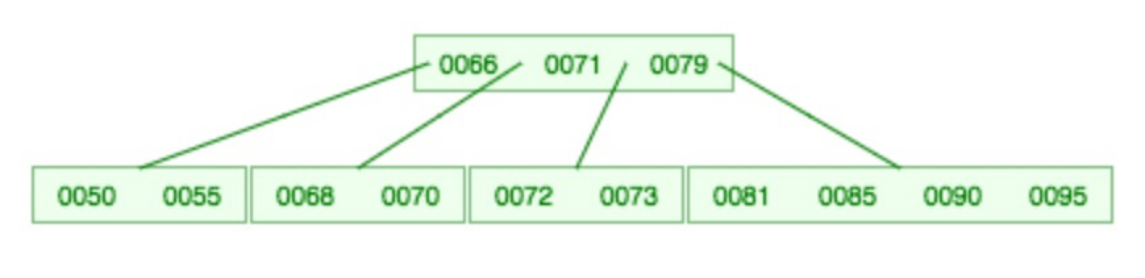
Exercises

- Show the results of entering the records with the keys in the order (50, 55, 66, 68, 70, 71, 72, 73, 79, 81, 85, 90, 95) to an initially empty B tree of order m = 5.

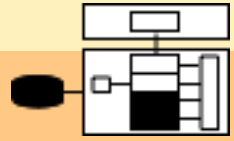


Exercises

- Show the results of entering the records with the keys in the order (50, 55, 66, 68, 70, 71, 72, 73, 79, 81, 85, 90, 95) to an initially empty B tree of order m = 5.

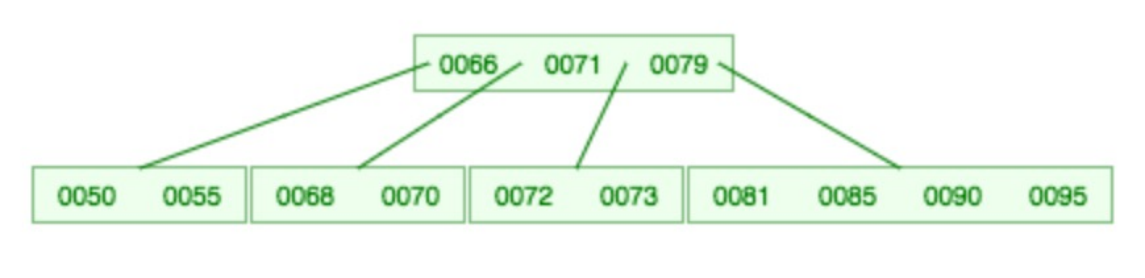


- What can you tell about the list of keys?

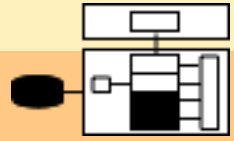


Exercises

- Show the results of entering the records with the keys in the order (50, 55, 66, 68, 70, 71, 72, 73, 79, 81, 85, 90, 95) to an initially empty B tree of order m = 5.

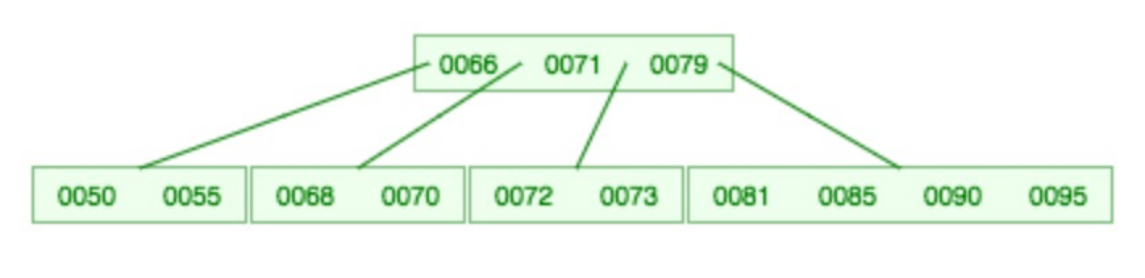


- What can you tell about the list of keys?
  - They are ordered

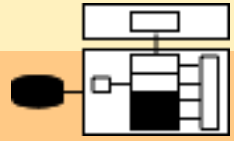


Exercises

- Show the results of entering the records with the keys in the order (50, 55, 66, 68, 70, 71, 72, 73, 79, 81, 85, 90, 95) to an initially empty B tree of order m = 5.

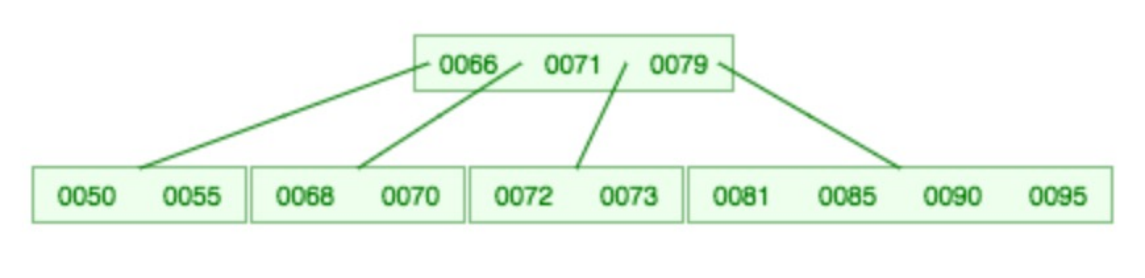


- What can you tell about the list of keys?
  - They are ordered
- What can you tell about the obtained tree?



Exercises

- Show the results of entering the records with the keys in the order (50, 55, 66, 68, 70, 71, 72, 73, 79, 81, 85, 90, 95) to an initially empty B tree of order m = 5.



- What can you tell about the list of keys?
  - They are ordered
- What can you tell about the obtained tree?
  - The B tree obtained have leave nodes that are filled at 50% with the exception of the last one.