B and B+ search tree

Marko Berezovský
Radek Mařík
PAL 2012

To read

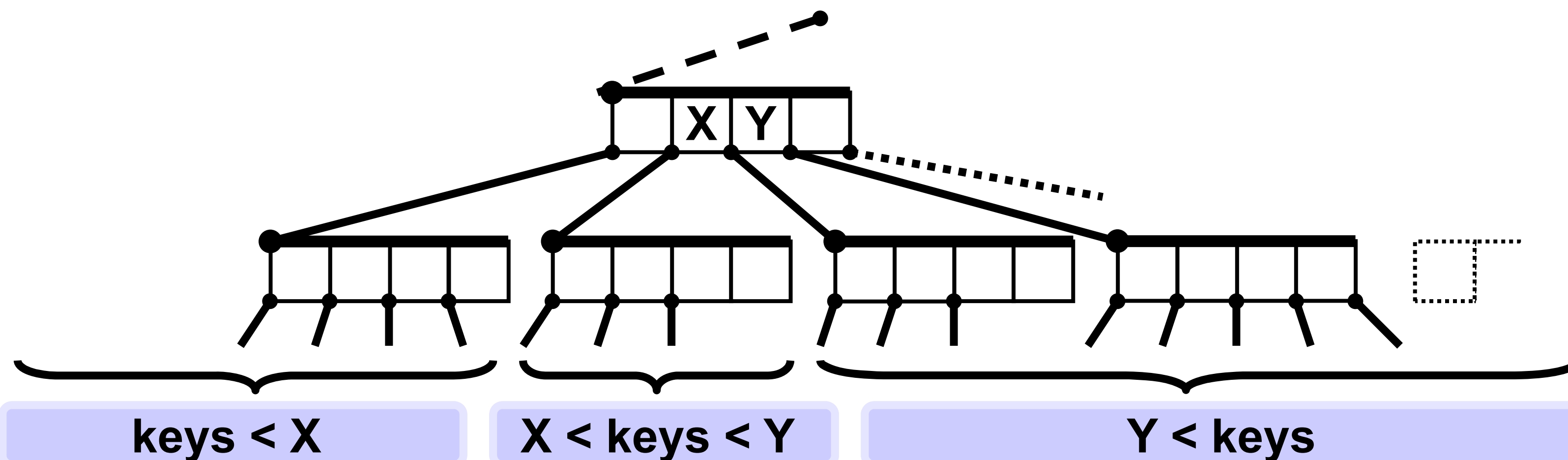
- Robert Sedgewick: *Algorithms in C++, Parts 1–4: Fundamentals, Data Structure, Sorting, Searching, Third Edition*, Addison Wesley Professional, 1998
- http://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf
- (CLRS) Cormen, Leiserson, Rivest, Stein: *Introduction to Algorithms*, 3rd ed., MIT Press, 2009

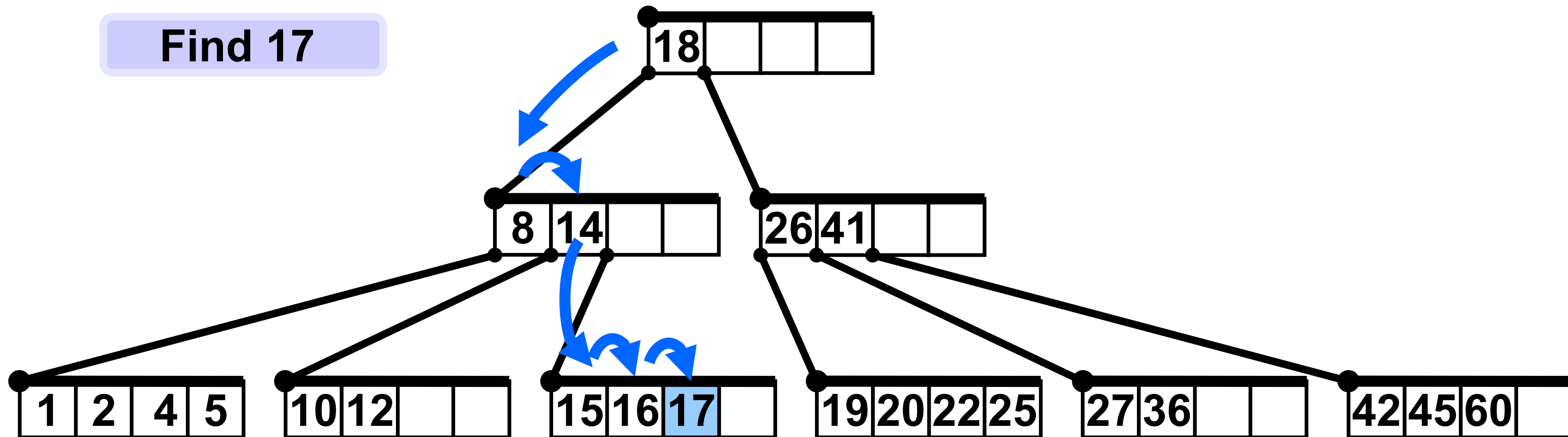
See PAL webpage for references

B tree

B-tree -- Rudolf Bayer, Edward M. McCreight, 1972

- All lengths of paths from the root to the leaves are equal.
- B-tree is perfectly balanced. Keys in the nodes are kept sorted.
- Fixed parameter k > 1 dictates the same size of all nodes.
- Each node except for the root contains at least k and at most 2k keys and if it is not a leaf it has at least k+1 and at most 2k+1 children.
- The root may contain any number of keys from 1 to 2k. If it is not simultaneously a leaf it has at least 2 and at most 2k+1children.



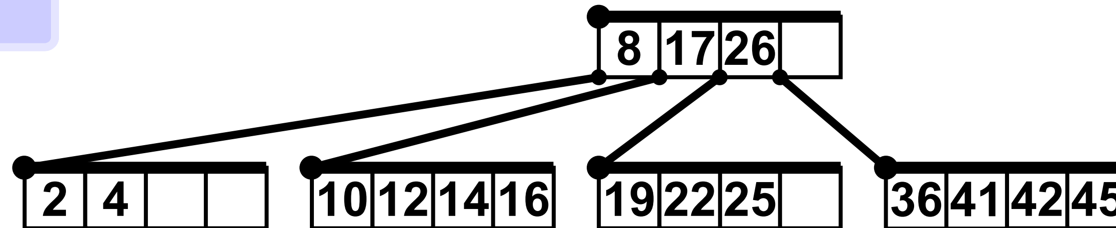


Search in the node is sequential (or binary or other...).

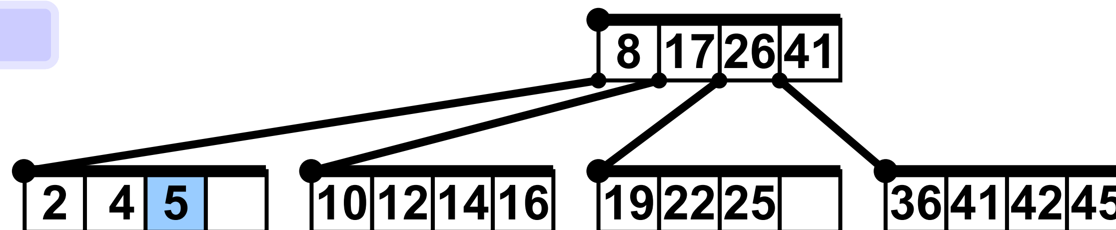
If the node is not a leaf and the key is not in the node
then the search continues in the appropriate child node.

If the node is a leaf and the key is not in the node
then the key is not in the tree.

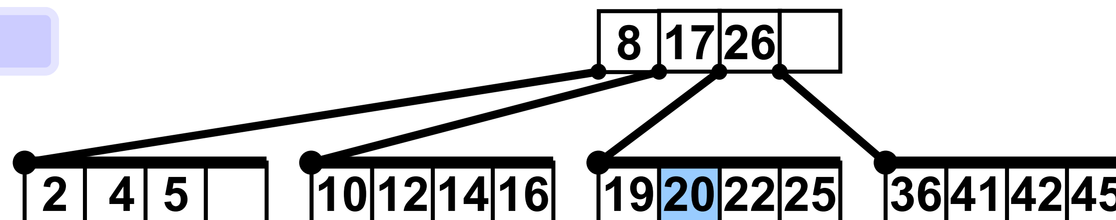
B-tree



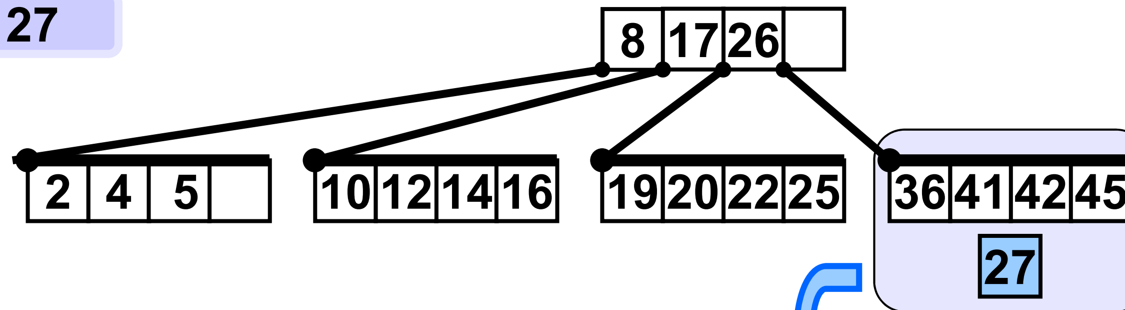
Insert 5



Insert 20

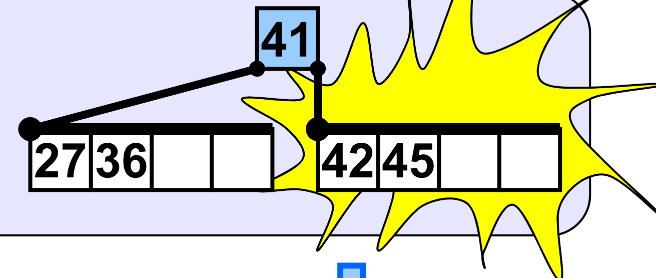


Insert 27

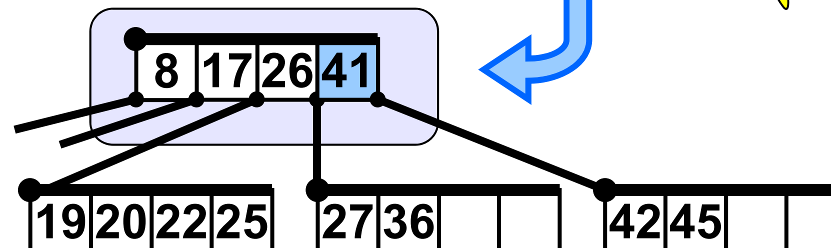


27 36 41 42 45

Sort outside the tree.



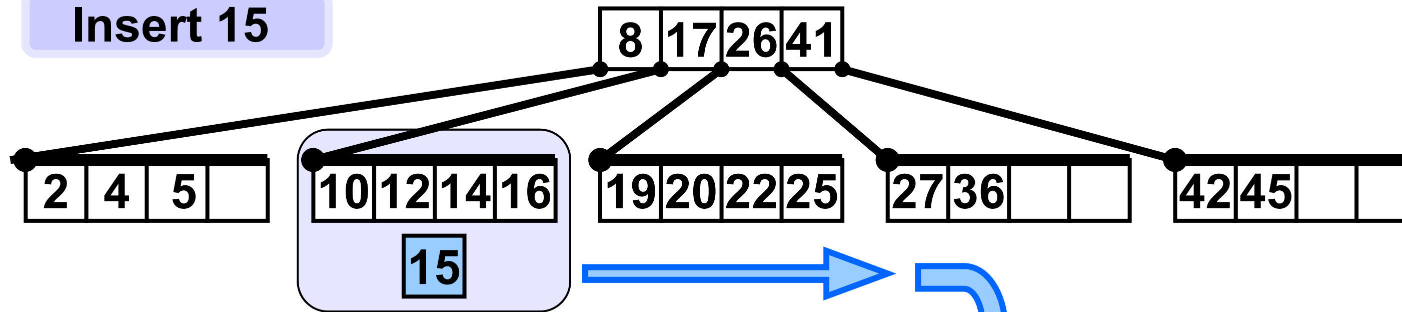
Select median,
create new node,
move to it the values
bigger than the median.



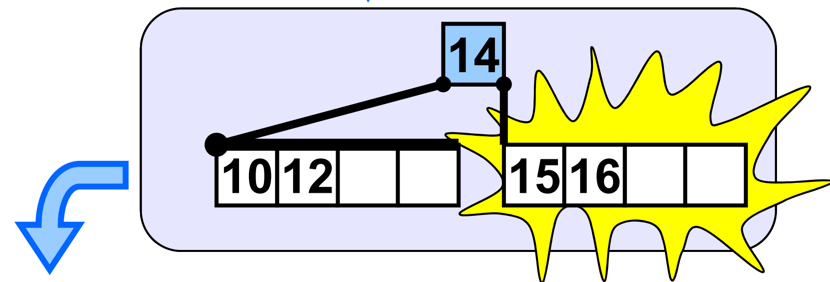
Try to insert the median
into the parent node.

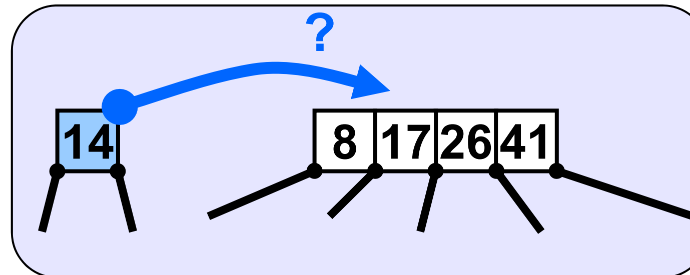
Success.

Insert 15



10 12 14 15 16





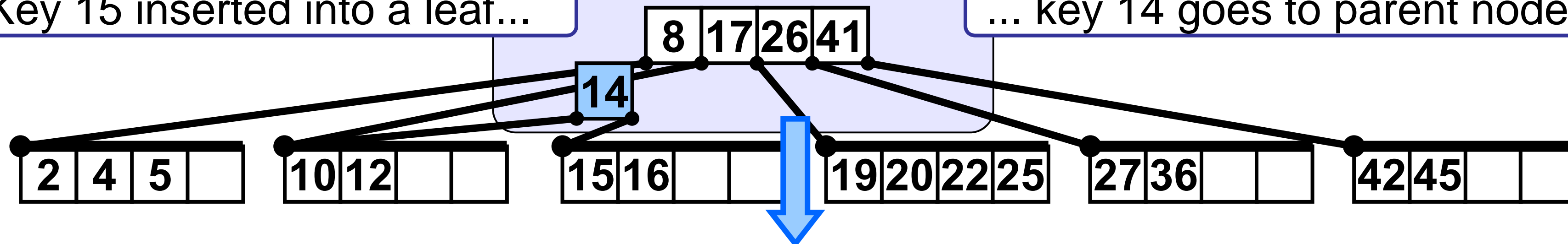
Sort outside the tree.

Select median,
create new node,
move to it the values
bigger than the median.

Try to insert the median
into the parent node.

Success?

B tree - Insert

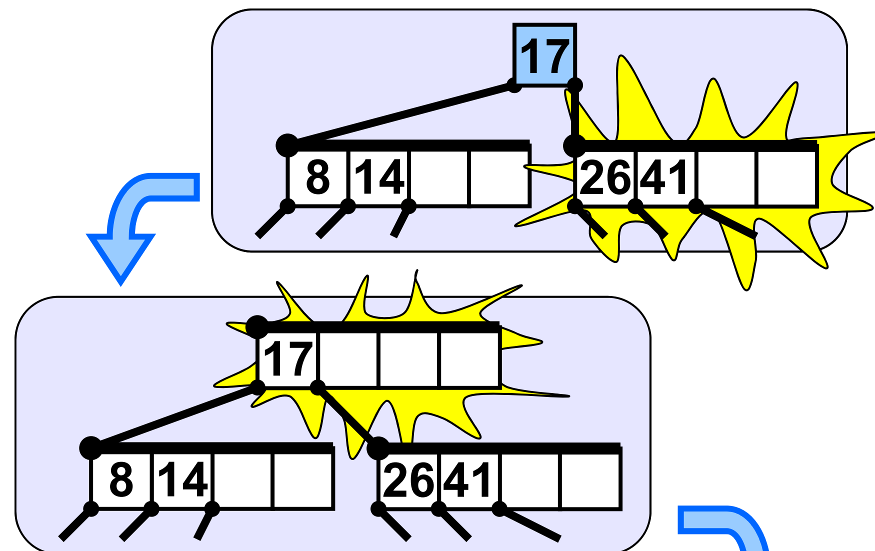


The parent node is full – repeat the process analogously.

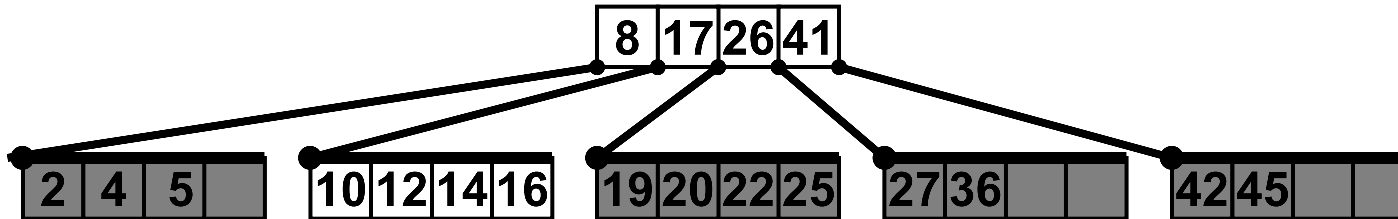
Sort values

Select median, create new node, move to it the values bigger than the median together with the corresponding references.

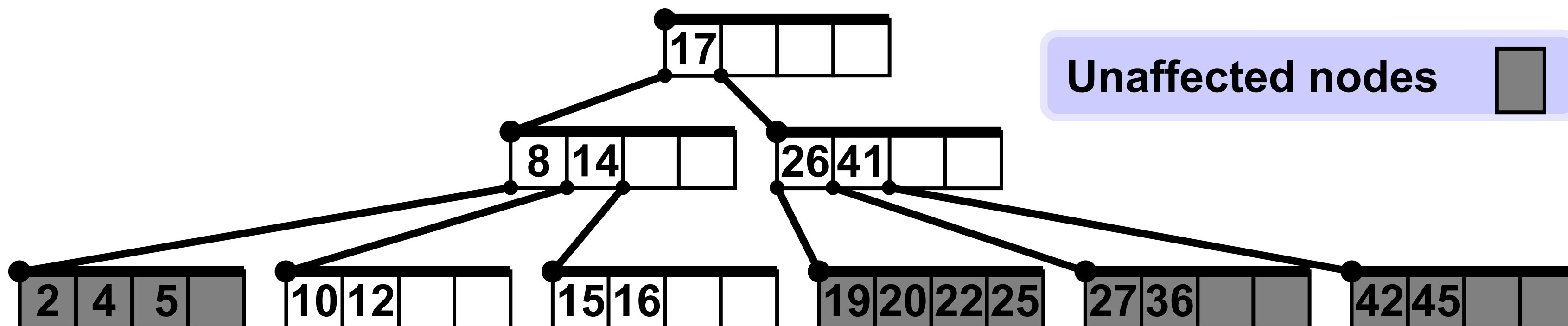
Cannot propagate the median into the parent (there is no parent), create a new root and store the median there.



Recapitulation - insert 15

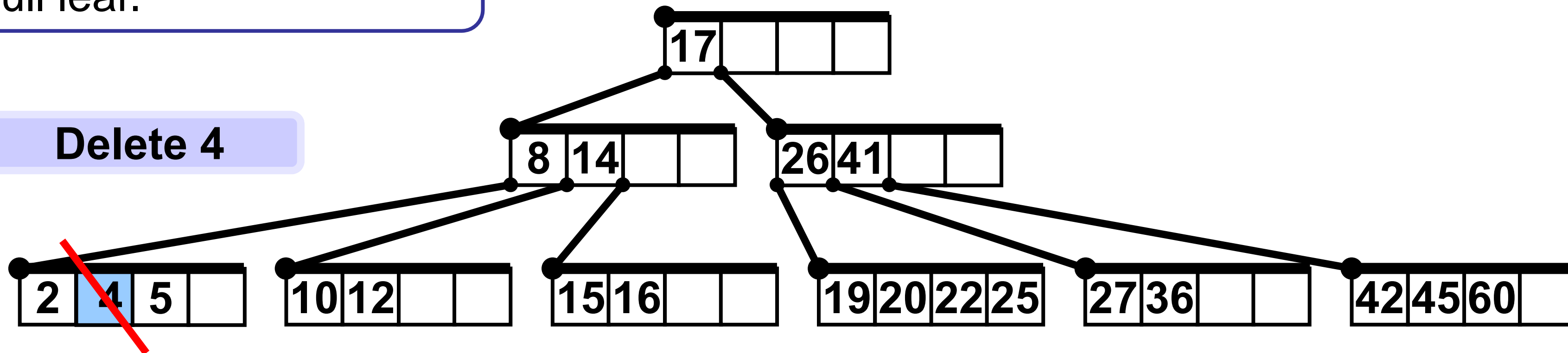


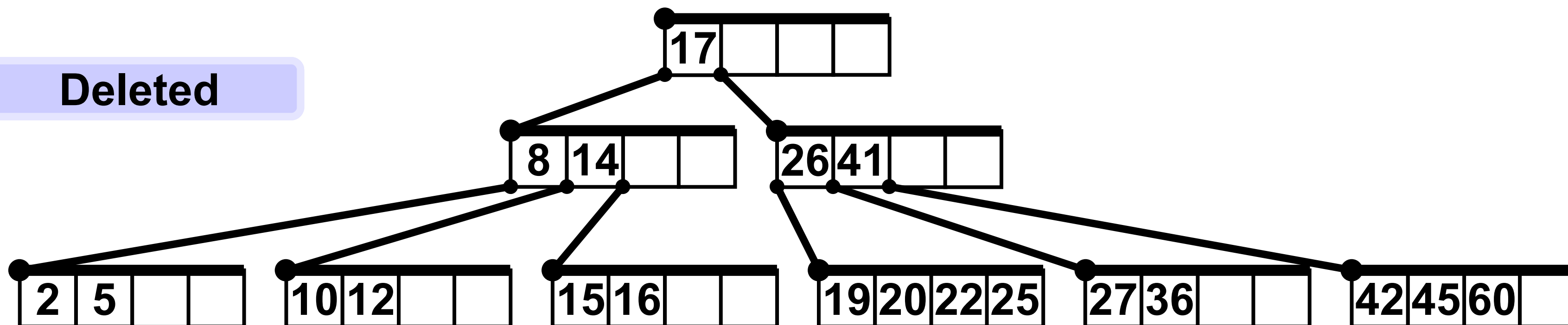
Insert 15



Each level acquired one new node, a new root was created too,
the tree grows upwards and remains perfectly balanced.

Delete in a sufficiently full leaf.

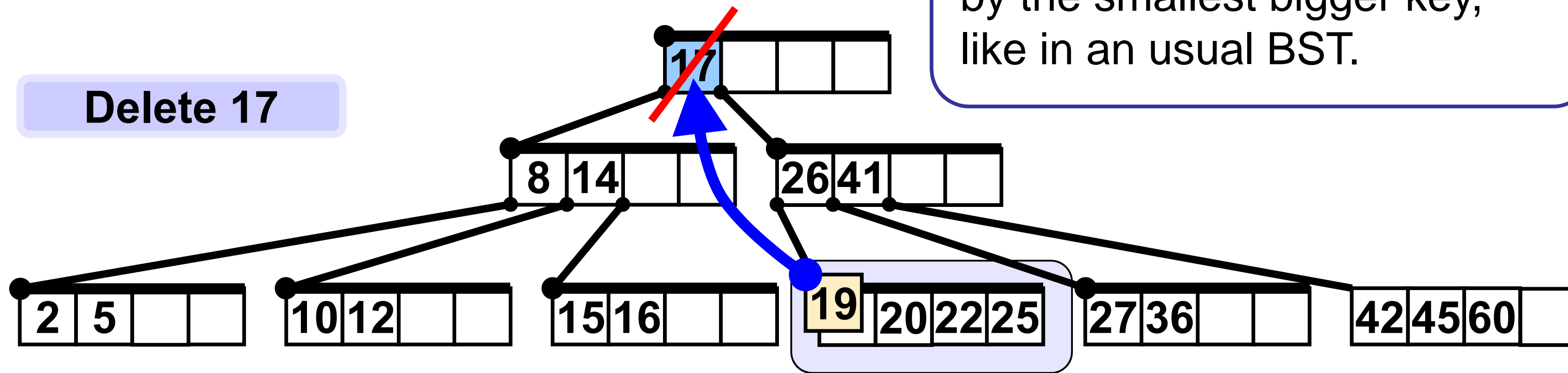




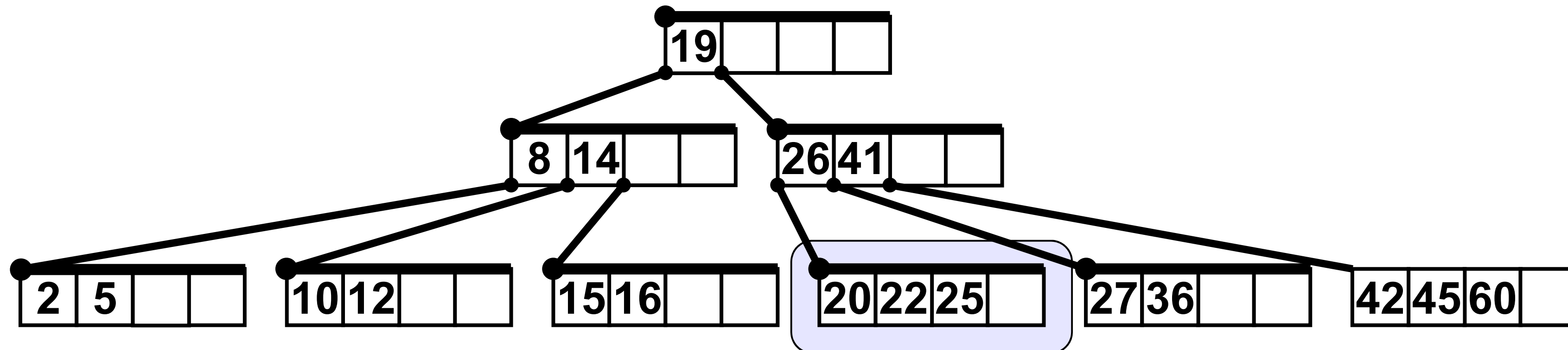
Delete in an internal node

The deleted key is substituted by the smallest bigger key, like in an usual BST.

Delete 17



The smallest bigger key is always in the leaf in a B-tree.
If the leaf is sufficiently full the delete operation is complete.

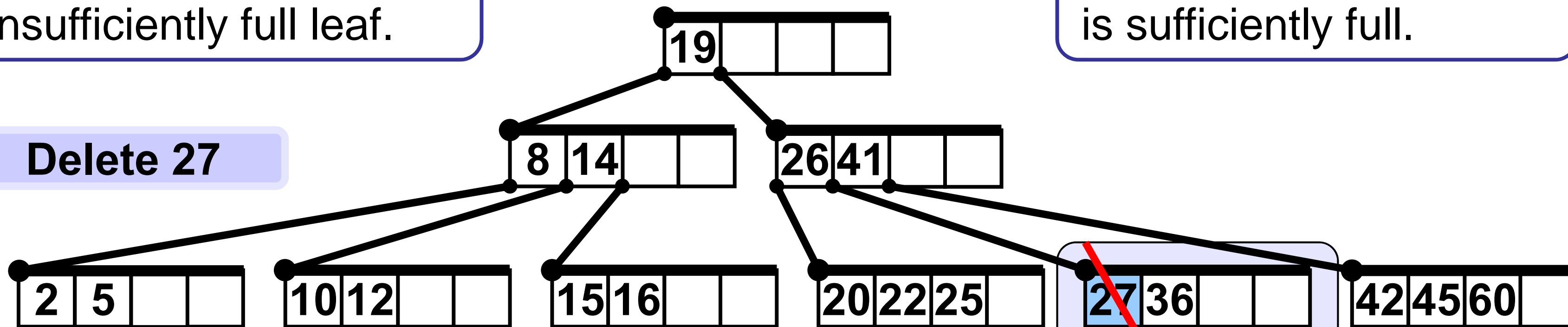


B tree - Delete

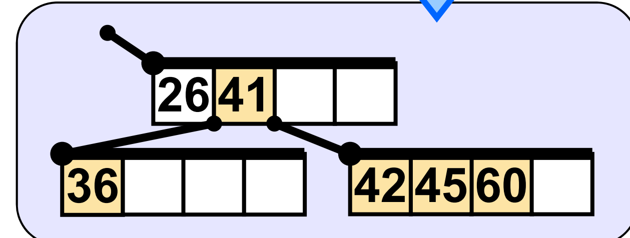
Delete in an insufficiently full leaf.

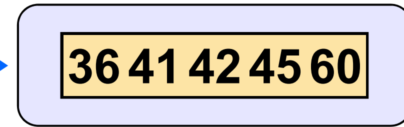
The neighbour leaf is sufficiently full.

Delete 27

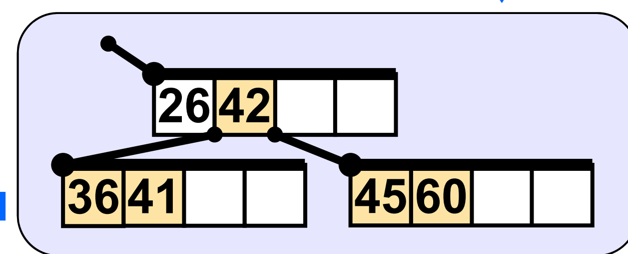


Merge the keys of the two leaves with the dividing key in the parent into one sorted list.

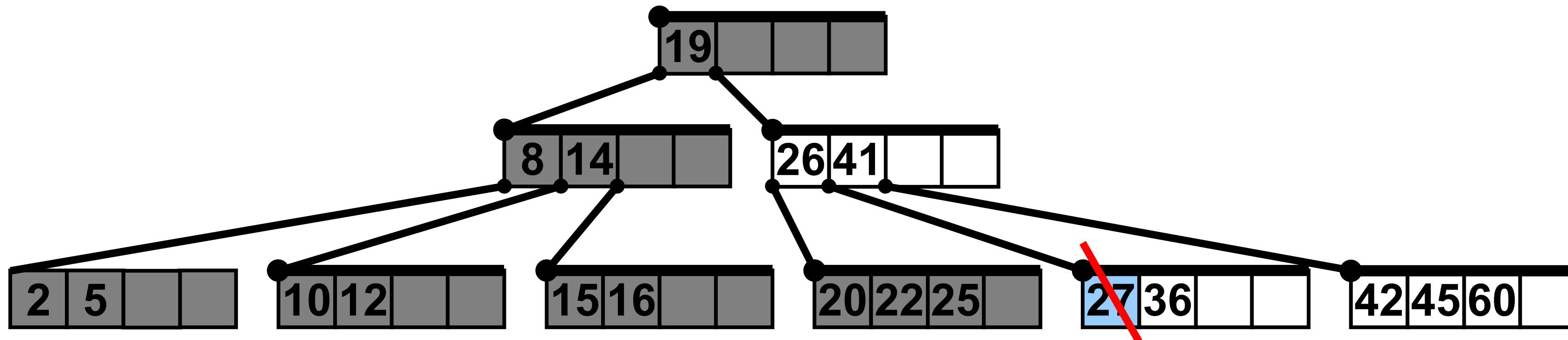




Insert the median of the sorted list into the parent and distribute the remainig keys into the left and right children of the median.

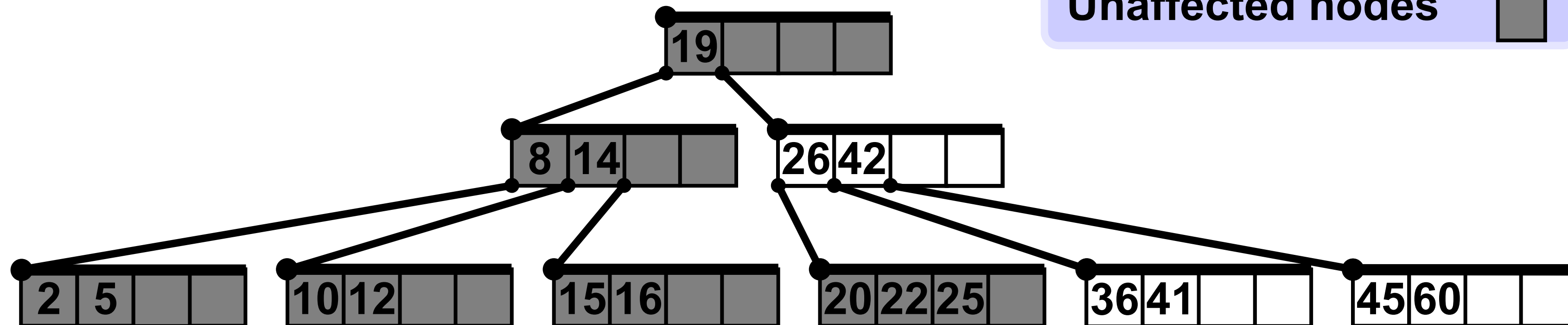


Recapitulation - delete 27



27 correctly deleted

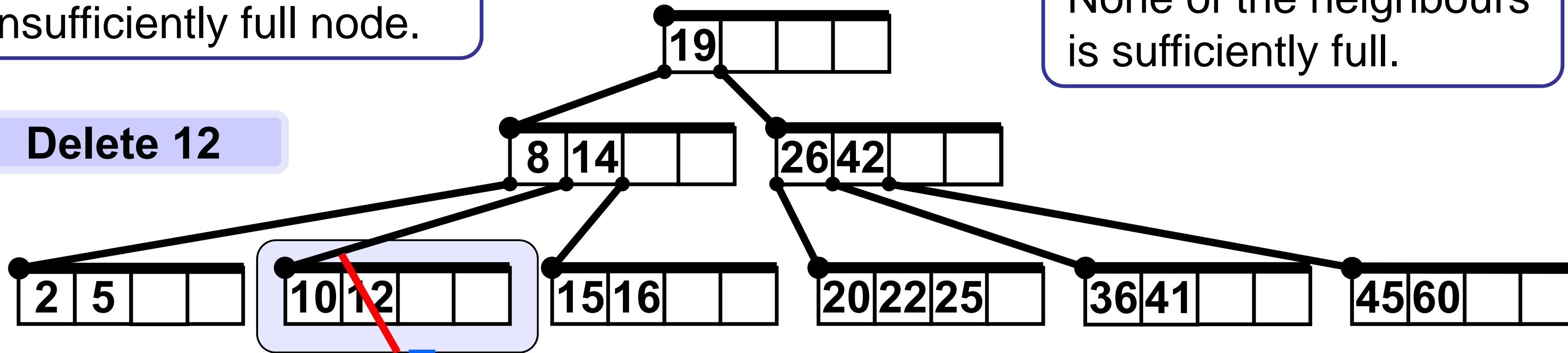
Unaffected nodes

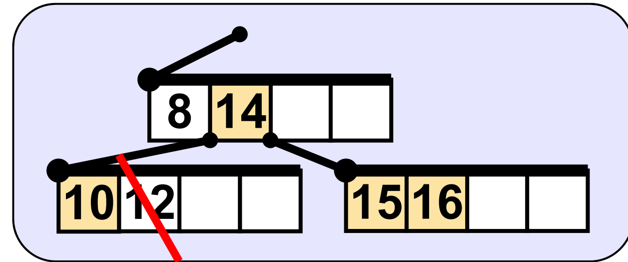


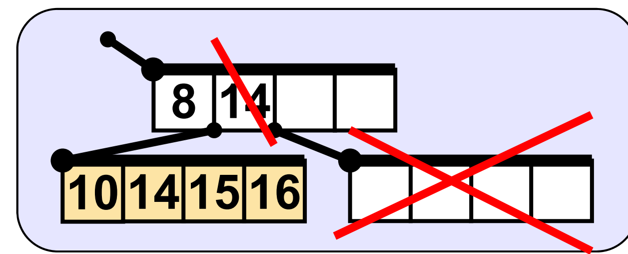
Delete in an insufficiently full node.

None of the neighbours is sufficiently full.

Delete 12

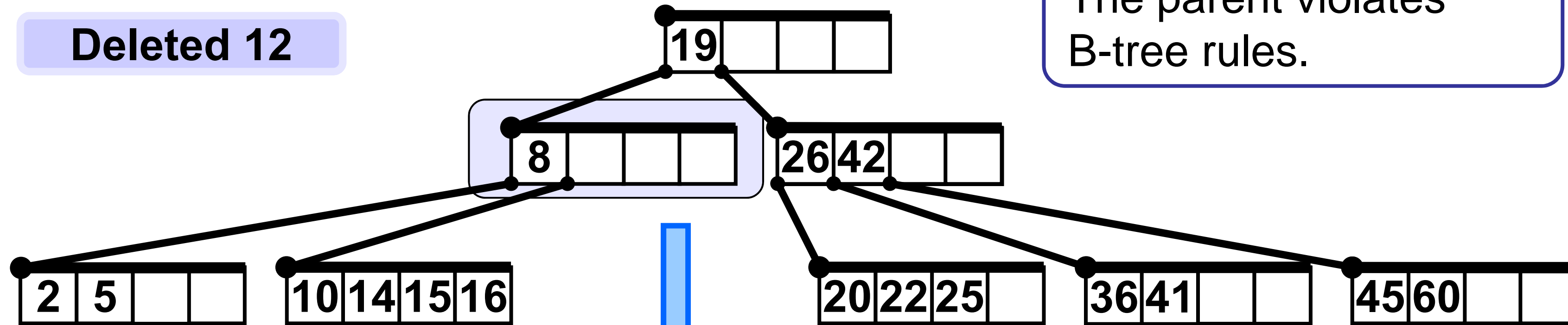




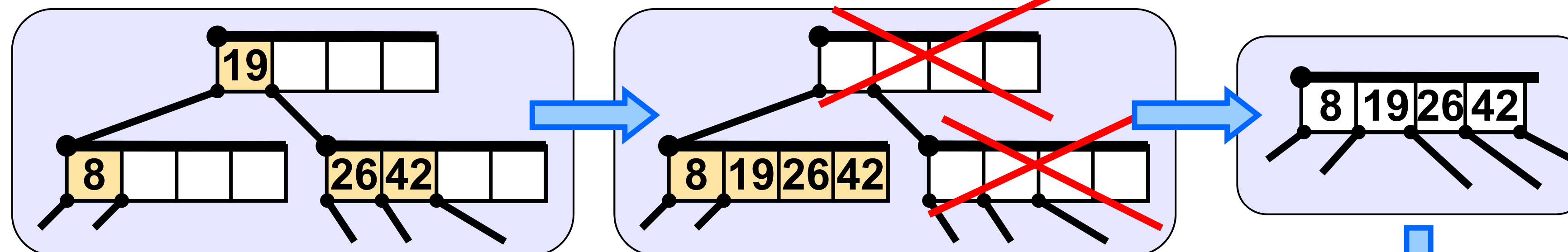


Merge the keys
of the node
and of one of the neighbours
and the median in the parent
into one sorted list.
Move all these keys to the original node, delete the neighbour, remove the original median and associated reference from the parent.

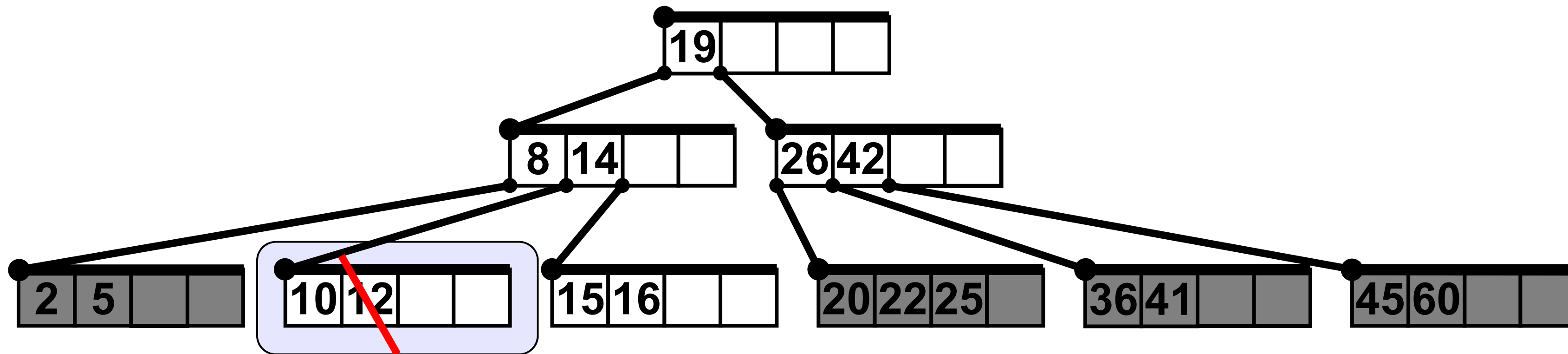
The parent violates B-tree rules.



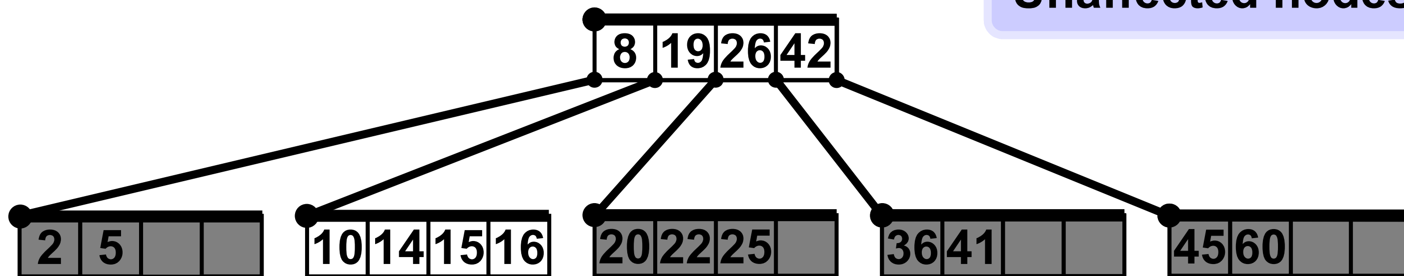
If the parent of the deleted node is not sufficiently full apply the same deleting strategy to the parent and continue the process towards the root until the rules of B-tree are satisfied.



Recapitulation - delete 12



Key 12 was deleted and the tree was reconstructed accordingly.

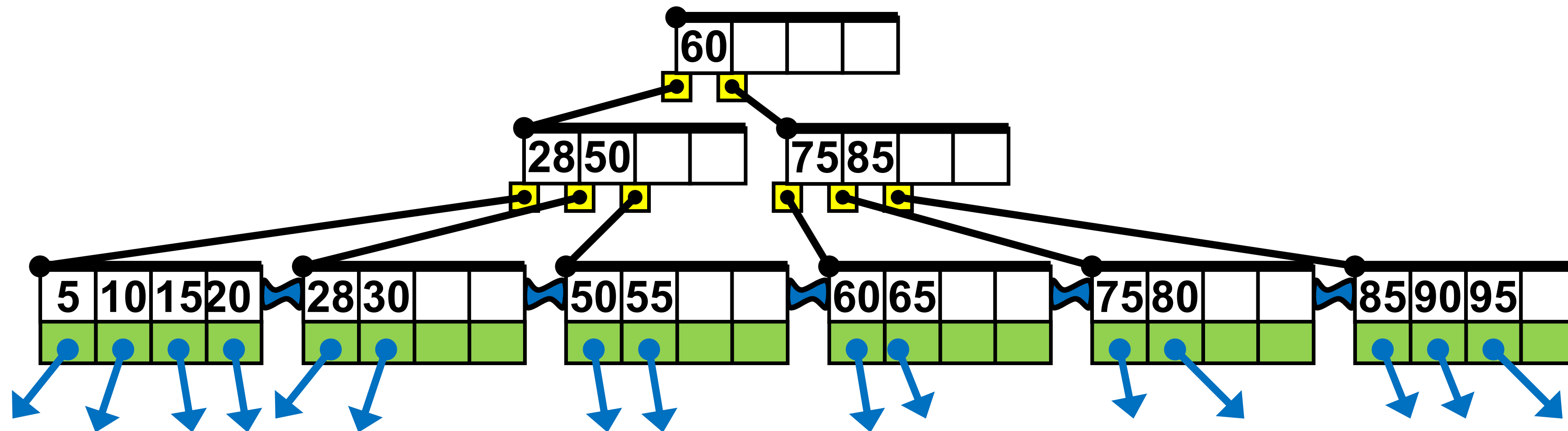


B+ tree

B+ tree is analogous to B-tree, namely in:
-- Being perfectly balanced all the time,
-- that nodes cannot be less than half full,
-- operational complexity.

The differences are:
-- Records (or pointers to actual records) are stored only in the leaf nodes,
-- internal nodes store only search key values which are used only as routers to guide the search.

The leaf nodes of a B^+-tree are linked together to form a linked list. This is done so that the records can be retrieved sequentially without accessing the B^+-tree index. This also supports fast processing of range-search queries.



Routers and keys 75

Data records
or pointers to them

Leaves links

Values in internal nodes are routers, originally each of them was a key when a record was inserted. Insert and Delete operations split and merge the nodes and thus move the keys and routers around. A router may remain in the tree even after the corresponding record and its key was deleted.

Values in the leaves are actual keys associated with the records and must be deleted when a record is deleted (their router copies may live on).

Inserting key K (and its associated data record) into B+ tree

Find, as in B tree, correct leaf to insert K. Then there are 3 cases:

Case 1

Free slot in a leaf? YES

Place the key and its associated record in the leaf.

Case 2

Free slot in a leaf? NO. Free slot in the parent node? YES.

1. Consider all keys in the leaf, including K, to be sorted.
2. Insert middle (median) key M in the parent node in the appropriate slot Y.
   (If parent does not exist, first create an empty one = new root.)
3. Split the leaf to two new leaves L1 and L2.
4. Left leaf (L1) from Y contains records with keys smaller than M.
5. Right leaf (L2) from Y contains records with keys equal to or greater than M.

Note: Splitting leaves and inner nodes works in the same way as in B-trees.

Inserting key K (and its associated data record) into B+ tree

Find, as in B tree, correct leaf to insert K. Then there are 3 cases:

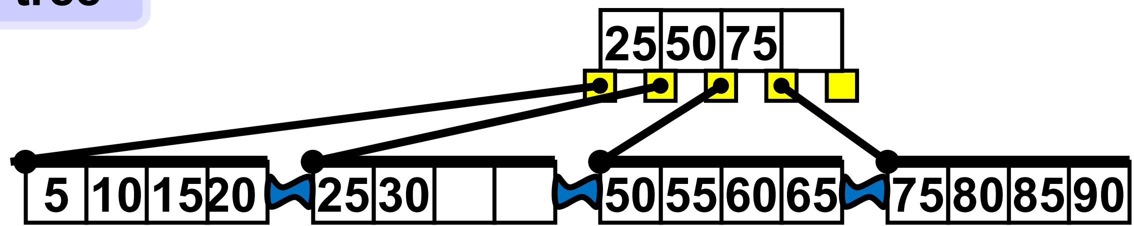
Case 3

Free slot in a leaf? NO. Free slot in the parent node? NO.

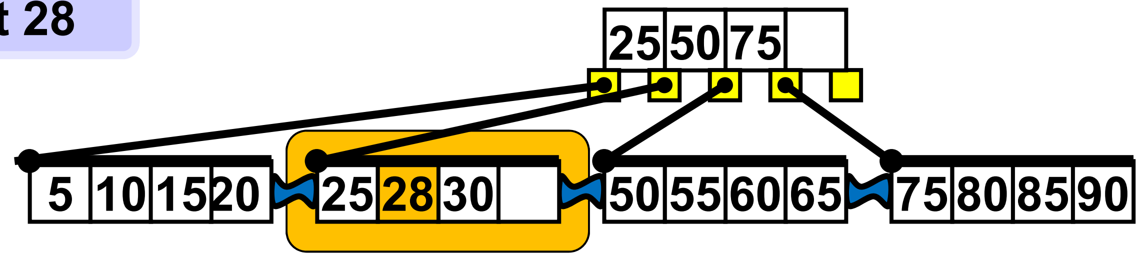
1. Split the leaf to two leaves L1 and L2, consider all its keys including K sorted, denote M median of these keys.
2. Records with keys < M go to the left leaf L1.
3. Records with keys >= M go to the right leaf L2.

4. Split the parent node P to nodes P1 and P2, consider all its keys including M sorted, denote M1 median of these keys.
5. Keys < M1 key go to P1.
6. Keys >= M1 key go to P2.
7. If parent PP of P is not full, insert M1 to PP and stop.
   (If PP does not exist, first create an empty one = new root.)
   Else set M := M1, P := PP and continue splitting parent nodes recursively up the tree, repeating from step 4.

Initial tree



Insert 28

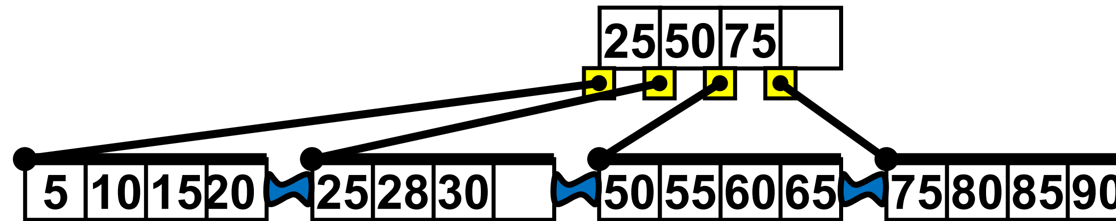


Changes

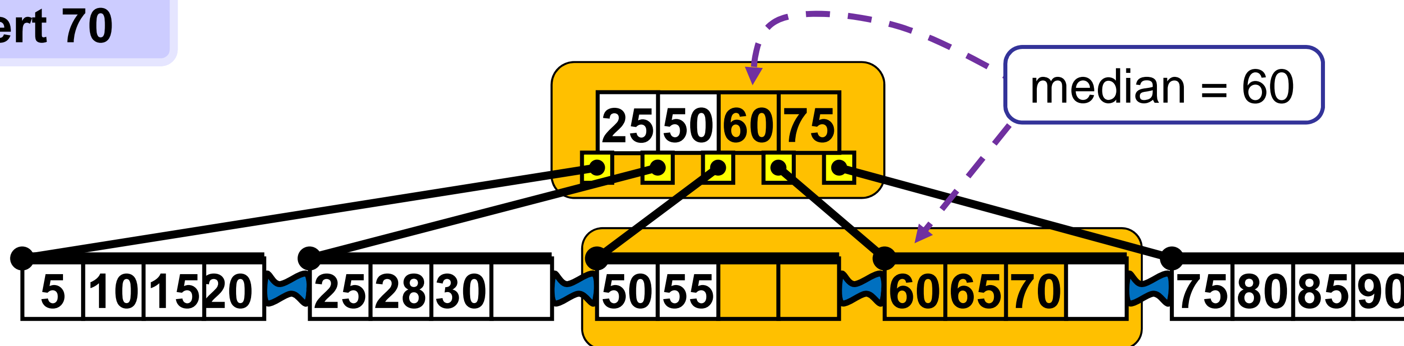
Leaves links

Data records and pointers to them are not drawn here for simplicity's sake.

Initial tree



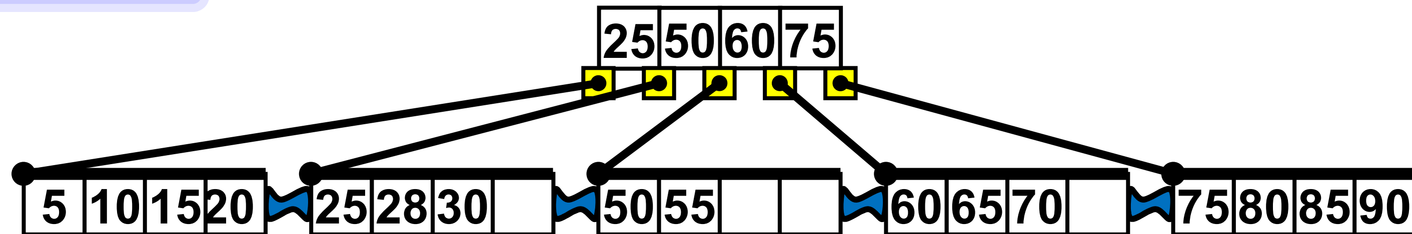
Insert 70



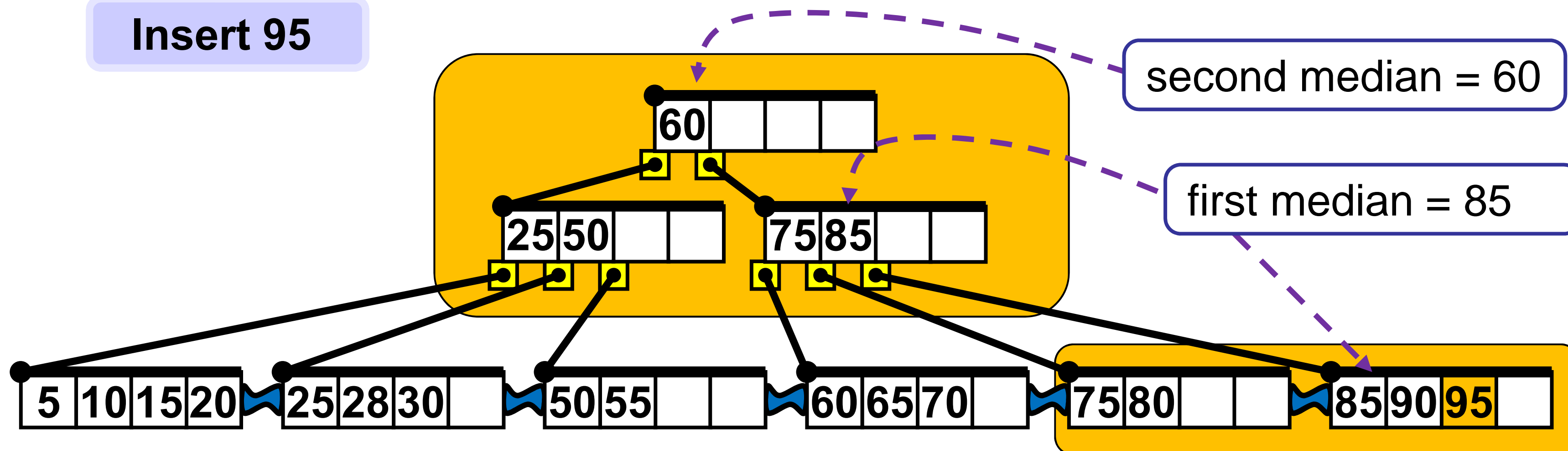
Changes

Leaves links

Initial tree



Insert 95



Changes

Leaves links

Note the router 60 in the root, detached from its original position in the leaf.

Deleting key K (and its associated data record) in B+ tree

Find, as in B tree, key K in a leaf. Then there are 3 cases:

Case 1

Leaf more than half full or leaf == root? YES.

Delete the key and its record from the leaf L. Arrange the keys in the leaf in ascending order to fill the void. If the deleted key K appears also in the parent node P replace it by the next bigger key K1 from L (explain why it exists) and leave K1 in L as well.

Case 2

Leaf more than half full? NO. Left or right sibling more than half full? YES.

Move one (or more if you wish and rules permit) key from sibling S to the leaf L, reflect the changes in the parent P of L and parent P2 of sibling S.
(If S does not exist then L is the root, which may contain any number of keys).

Deleting key K (and its associated data record) in B+ tree

Find, as in B tree, key K in a leaf. Then there are 3 cases:

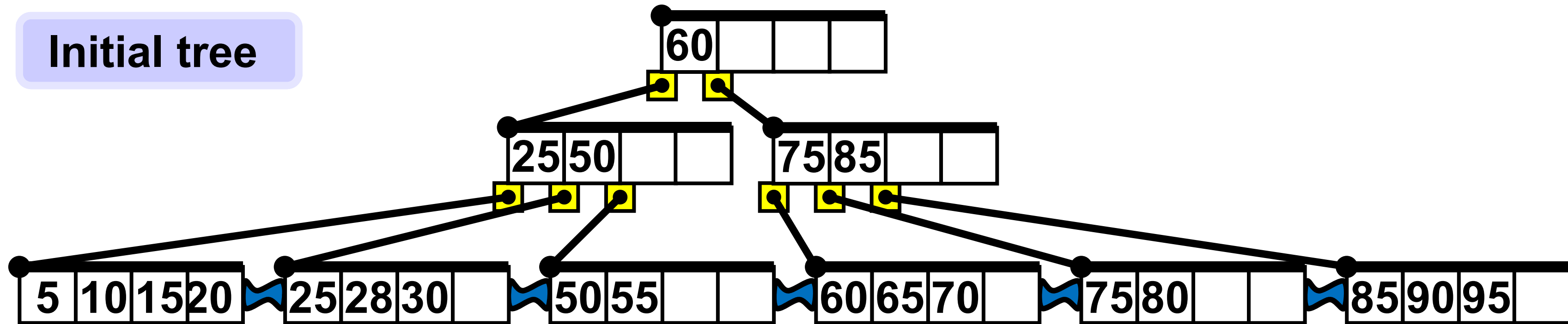
Case 3

Leaf more than half full? NO. Left or right sibling more than half full? NO.

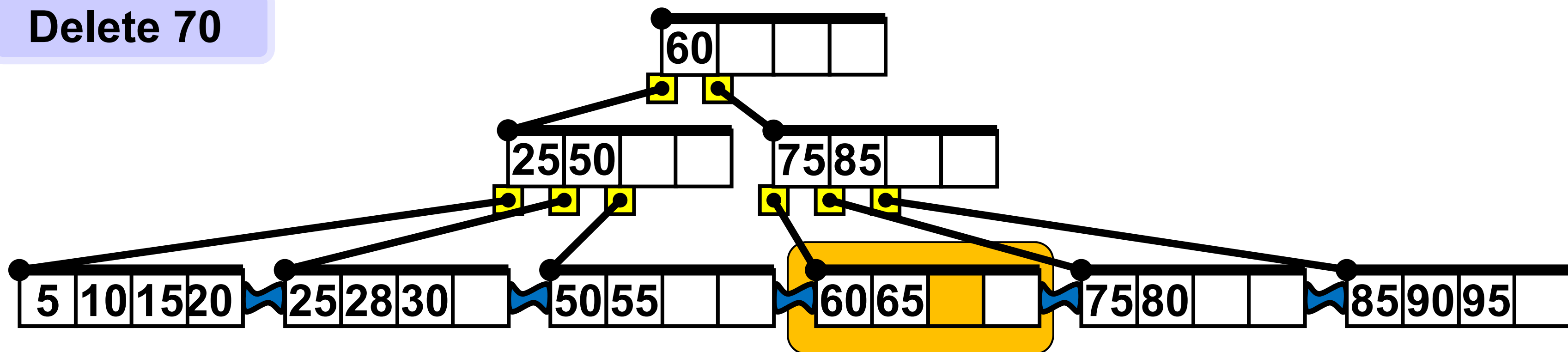
1. Consider sibling S of L which has same parent P as L.
2. Consider set M of ordered keys of L and S without K but together with key K1 in P which separates L and S.
3. Merge: Store M in L, connect L to the other sibling of S (if exists), destroy S.
4. Set the reference left to K1 to point to L. Delete K1 from P. If P contains K delete it also from P. If P is still at least half full stop, else continue with 5.
5. If any sibling SP of P is more then half full, move necessary number of keys from SP to P and adjust links in P, SP and their parents accordingly and stop. Else set L := P and continue recursively up the tree (like in B-tree), repeating from step 1.

Note: Merging leaves and inner nodes works the same way as in B-trees.

Initial tree



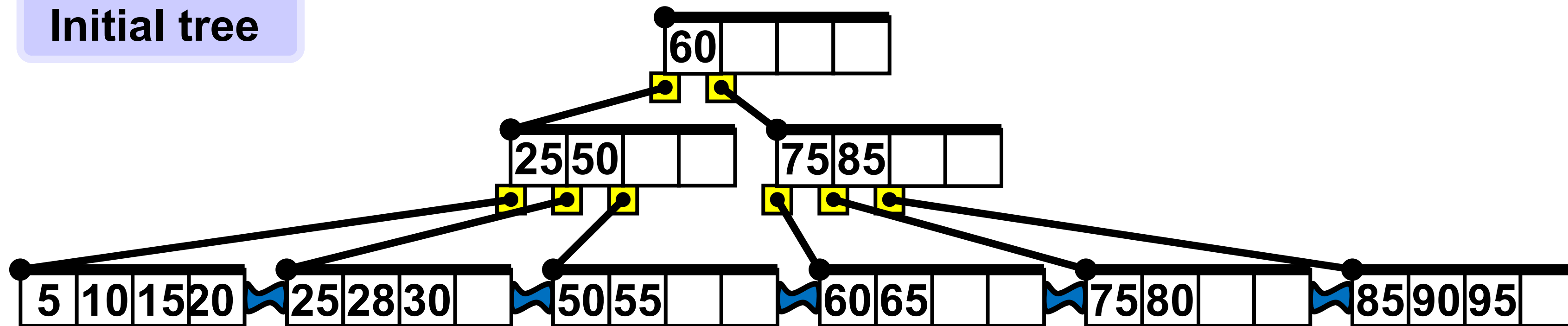
Delete 70



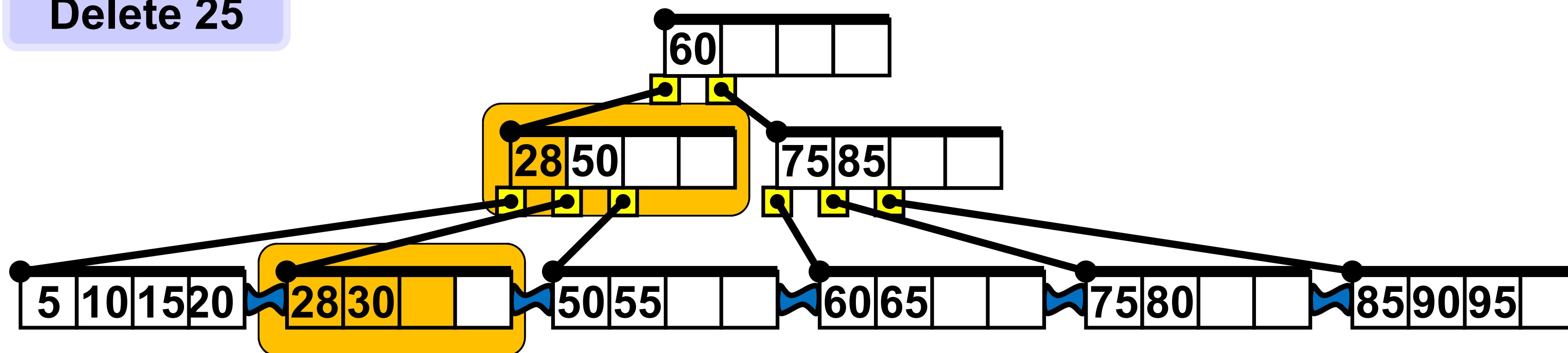
Changes

Leaves links

Initial tree

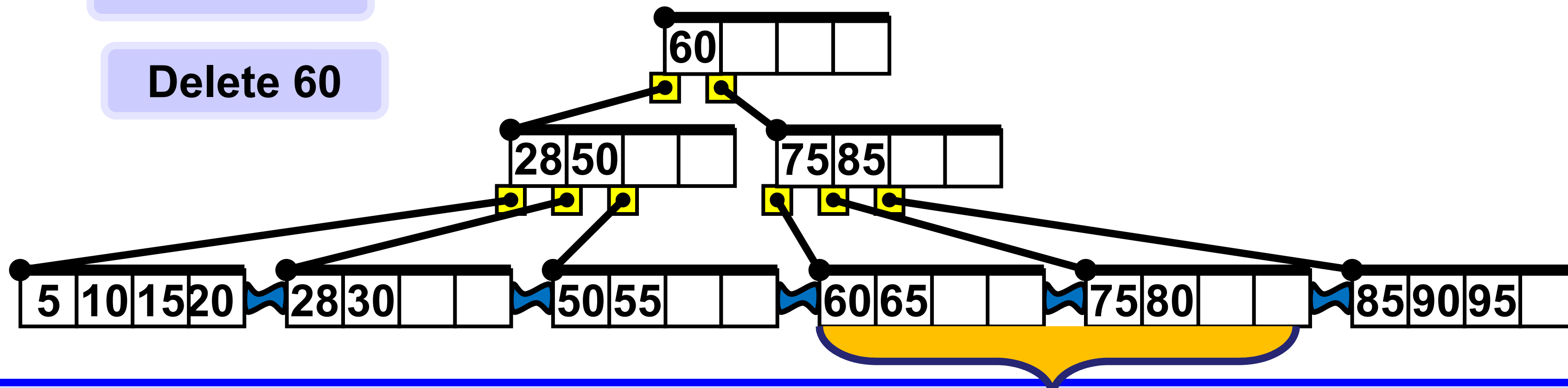


Delete 25

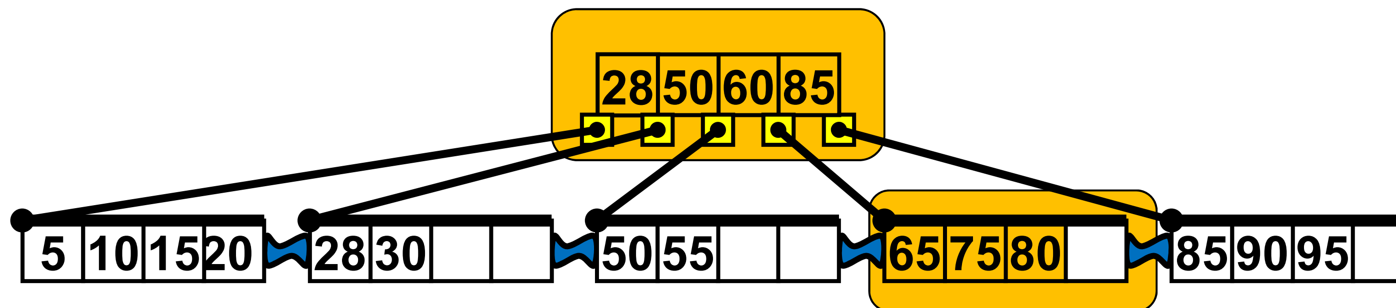


Changes

Leaves links

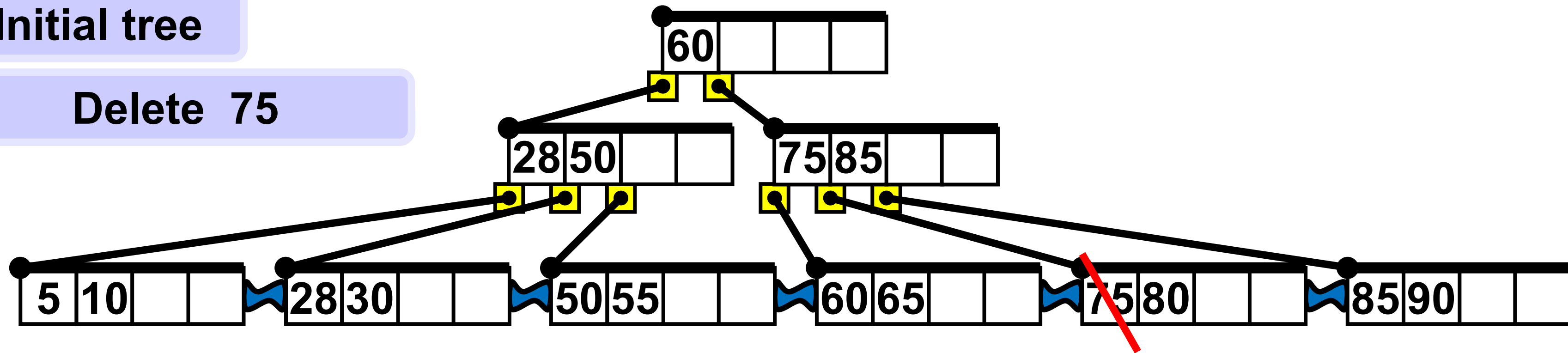


Merge



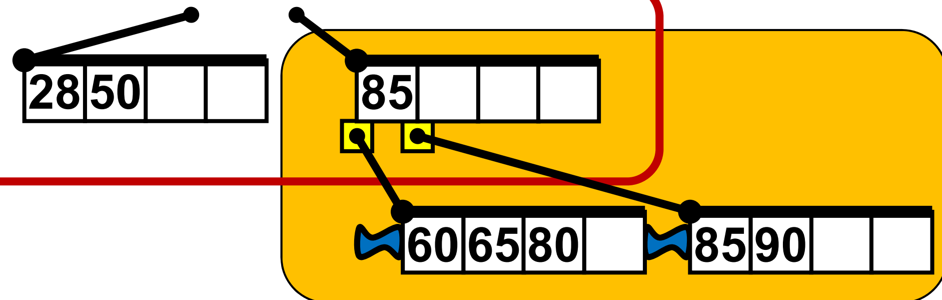
Initial tree

Delete 75

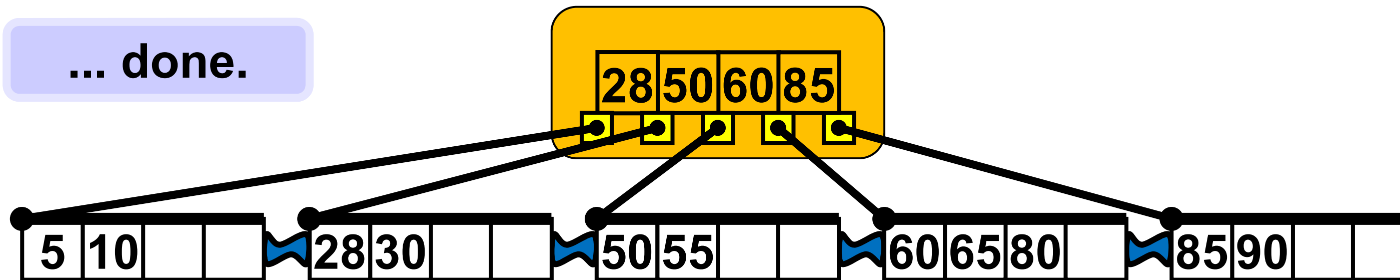


Too few keys, merge these two nodes and bring key from parent (recursively).

Progress...



... done.



Complexities

Find, Insert, Delete,
all need $\Theta(b \log_b n)$ operations, where n is number of records in the tree,
and b is the branching factor or, as it is often understood, the order of the tree.

Note: Be careful, some authors (e.g CLRS) define degree/order of B-tree as [b/2], there is no unified precise common terminology.

Range search thanks to the linked leaves is performed in time
$\Theta(b \log_b(n) + k/b)$
where k is the range (number of elements) of the query.