Annotation

- AVL-trees,
- red-black-trees,
- A-B-trees.

Balancedness

- Generally, it is an unpleasant problem.

Balancedness

- Generally, it is an unpleasant problem.
- Thus AVL-trees got introduced with a bit relaxed notion of balancedness.

Balancedness

- Generally, it is an unpleasant problem.
- Thus AVL-trees got introduced with a bit relaxed notion of balancedness.
- AVL-tree is a BST where for each element the depth of the left subtree differs at most by 1 from the depth of the right subtree.

Balancedness

- Generally, it is an unpleasant problem.
- Thus AVL-trees got introduced with a bit relaxed notion of balancedness.
- AVL-tree is a BST where for each element the depth of the left subtree differs at most by 1 from the depth of the right subtree.
- AVL – Adelson-Velskij and Landis.

Balancedness

- Generally, it is an unpleasant problem.
- Thus AVL-trees got introduced with a bit relaxed notion of balancedness.
- AVL-tree is a BST where for each element the depth of the left subtree differs at most by 1 from the depth of the right subtree.
- AVL – Adelson-Velskij and Landis.
- Operations `member`, `insert` and `delete` are the same as for BST, just

Balancedness

- Generally, it is an unpleasant problem.
- Thus AVL-trees got introduced with a bit relaxed notion of balancedness.
- AVL-tree is a BST where for each element the depth of the left subtree differs at most by 1 from the depth of the right subtree.
- AVL – Adelson-Velskij and Landis.
- Operations `member`, `insert` and `delete` are the same as for BST, just
- after `insert` and `delete` we perform the balance-renewing operations.

Balancedness

- Generally, it is an unpleasant problem.
- Thus AVL-trees got introduced with a bit relaxed notion of balancedness.
- AVL-tree is a BST where for each element the depth of the left subtree differs at most by 1 from the depth of the right subtree.
- AVL – Adelson-Velskij and Landis.
- Operations `member`, `insert` and `delete` are the same as for BST, just
- after `insert` and `delete` we perform the balance-renewing operations.
- For each vertex we define a value `balance` saying `depth_right - depth_left`, permitted values are `-1, 0` and `1`.

Balance-renewing operations

- Problem appears with balance WLOG 2.

Balance-renewing operations

- Problem appears with balance WLOG 2.
- We start solving on the bottom-most level with this balance.

Balance-renewing operations

- Problem appears with balance WLOG 2.
- We start solving on the bottom-most level with this balance.
- We explore two possibilities, the remaining 2 are symmetric.

Balance-renewing operations

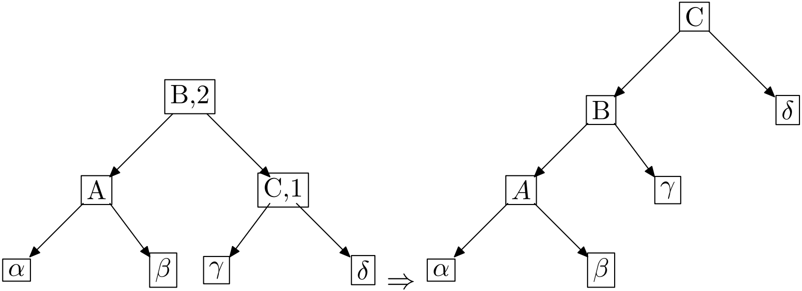
- Problem appears with balance WLOG 2.
- We start solving on the bottom-most level with this balance.
- We explore two possibilities, the remaining 2 are symmetric.
- The tree may be falling "to the side" or "to the interior".

Balance-renewing operations

- Problem appears with balance WLOG 2.
- We start solving on the bottom-most level with this balance.
- We explore two possibilities, the remaining 2 are symmetric.
- The tree may be falling "to the side" or "to the interior".
- In the former case we use a rotation, in the latter a double-rotation.

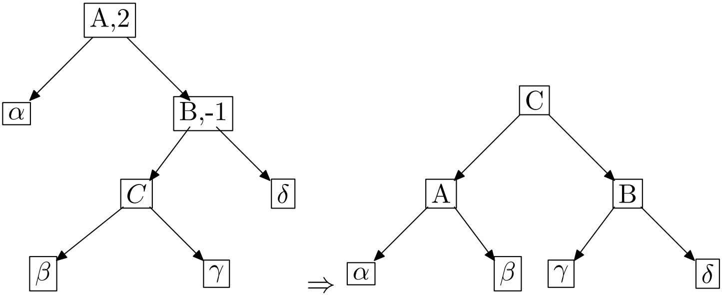
Rotation

Tree is falling "to the side".



Double-rotation

Tree is falling "to the interior".



Analysis and remarks

rotation, double-rotation, depths

- While inserting, one rotation (or double-rotation) suffices.

Analysis and remarks

rotation, double-rotation, depths

- While inserting, one rotation (or double-rotation) suffices.
- Delete may start a cascade of rotations (the distortion is travelling towards the root).

Analysis and remarks

rotation, double-rotation, depths

- While inserting, one rotation (or double-rotation) suffices.
- Delete may start a cascade of rotations (the distortion is travelling towards the root).
- Number of elements in an AVL-tree with depth n:

Analysis and remarks

rotation, double-rotation, depths

- While inserting, one rotation (or double-rotation) suffices.
- Delete may start a cascade of rotations (the distortion is travelling towards the root).
- Number of elements in an AVL-tree with depth n:
- Depth of the sons differs at most by one, thus: $T(n) \geq T(n-1) + T(n-2)$,

Analysis and remarks

rotation, double-rotation, depths

- While inserting, one rotation (or double-rotation) suffices.
- Delete may start a cascade of rotations (the distortion is travelling towards the root).
- Number of elements in an AVL-tree with depth n:
- Depth of the sons differs at most by one, thus: $T(n) \geq T(n-1) + T(n-2)$,
- Thus the number of elements is at least the nth Fibonacci number,

Analysis and remarks

rotation, double-rotation, depths

- While inserting, one rotation (or double-rotation) suffices.
- Delete may start a cascade of rotations (the distortion is travelling towards the root).
- Number of elements in an AVL-tree with depth n:
- Depth of the sons differs at most by one, thus: $T(n) \geq T(n-1) + T(n-2)$,
- Thus the number of elements is at least the nth Fibonacci number,
- thus the depth is logarithmic w.r.t. number of elements.

Red-black trees

- Another method how to keep the tree sufficiently spreaded.

Red-black trees

- Another method how to keep the tree sufficiently spreaded.
- Each vertex is colored with red or black color.

Red-black trees

- Another method how to keep the tree sufficiently spreaded.
- Each vertex is colored with red or black color.
- Red vertices must not appear one after another,

Red-black trees

- Another method how to keep the tree sufficiently spreaded.
- Each vertex is colored with red or black color.
- Red vertices must not appear one after another,
- number of black vertices is the same for any path from the root to all the leaves.

Red-black trees

- Another method how to keep the tree sufficiently spreaded.
- Each vertex is colored with red or black color.
- Red vertices must not appear one after another,
- number of black vertices is the same for any path from the root to all the leaves.
- Thus one subtree has depth at most twice larger than the other.

Red-black trees

- Another method how to keep the tree sufficiently spreaded.
- Each vertex is colored with red or black color.
- Red vertices must not appear one after another,
- number of black vertices is the same for any path from the root to all the leaves.
- Thus one subtree has depth at most twice larger than the other.
- The tree is administrated using rotations, double-rotations and recoloring.

Red-black trees

- Another method how to keep the tree sufficiently spreaded.
- Each vertex is colored with red or black color.
- Red vertices must not appear one after another,
- number of black vertices is the same for any path from the root to all the leaves.
- Thus one subtree has depth at most twice larger than the other.
- The tree is administrated using rotations, double-rotations and recoloring.
- Exact rules get lectured on Algorithms.

Red-black trees

- Another method how to keep the tree sufficiently spreaded.
- Each vertex is colored with red or black color.
- Red vertices must not appear one after another,
- number of black vertices is the same for any path from the root to all the leaves.
- Thus one subtree has depth at most twice larger than the other.
- The tree is administrated using rotations, double-rotations and recoloring.
- Exact rules get lectured on Algorithms.
- The depth is also logarithmic w.r.t. number of elements.

Shown on a blackboard

A-B-trees, *k*-ary tree canonical representation.