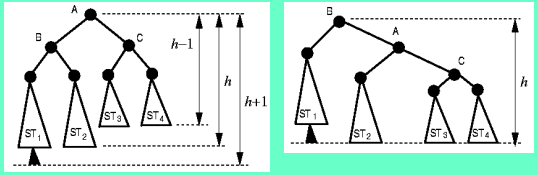
Balancing an AVLTree

- Two mirror-symmetric pairs of cases to rebalance the tree if after the insertion of a new key to the bottom the AVL property is invalidated
- Only one single or double rotation is sufficient
- Deletions are more complicated: $O(\log n)$ rotations can be required to restore the balance
- AVL balancing is not computationally efficient . Better balanced search trees: red-black, AA-trees, B-trees

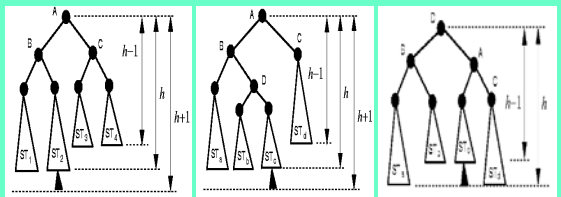
Single Rotation

- The inserted new key invalidates the AVL property
- To restore the AVL tree, the root is moved to the node B and the rest of the tree is reorganised as the BST



Double Rotation

- Balancing by more complex alternative representation of the tree (split of the ST2 and move of the root to D)



Red-Black Tree

- A **red-black tree** is a binary search tree with the following ordering properties:
  - Every node is coloured either **red** or **black**
  - The root is **black**
  - In a node is **red**, its children must be **black**
  - Every path from a node to a leaf must contain the same number of **black** nodes
- Statement: because every path from the root to a leaf contains b black nodes, there are at least 2^b black nodes in the tree

Red-Black Tree

- Proof of the statement by math induction:

  it is valid for $b=1$ (only the root or also1-2 its children)
- Let it be valid for all red-black trees with b black nodes per path
- If a tree contains $b+1$ black nodes per path and it has 2 black children of the root, then it contains at least

$$2\cdot(2^b-1)+1 = 2^{b+1}-1 \text{ black nodes}$$

Red-Black Tree

- If there is a red child of the root, it has necessarily only black children, and the total number of the black nodes is even larger
- There cannot be two consecutive red nodes on a path, so the height of a red-black tree is at most $2\log_2(n+1)$
- Thus the search operation is logarithmic $O(\log n)$

Red-Black Tree

Average case: a search in a red-black tree with n nodes built from random keys seems to require about $\log\ n$ comparisons and an insertion seems to require, on the average, less than one rotation

Worst case: a search in a red-black tree with n nodes built from random keys requires less than $2\log\ n + 2$ comparisons, and an insertion requires fewer than one-quarter as many rotations as comparisons

AA-Trees

- Software implementation of the operations insert and remove for red-black trees is a rather tricky process
- A balanced search **AA-tree** is a method of choice if deletions are needed
- The AA-tree adds one extra condition to the **red-black** tree: left children may not be red
- This condition greatly simplifies the red-black tree remove operation

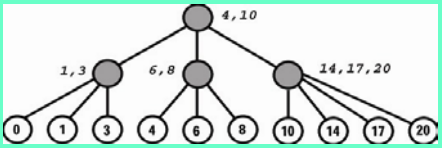
B-Trees: Efficient External Search

- For very big databases, even $\log_2 n$ search steps may be unacceptable
- To reduce the number of disk accesses: an optimal m-ary search tree of height about $\log_m n$

m-way branching lowers the optimal tree height by factor $\log_2 m$ (i.e., by 3.3 if m=10)

n	10^5	10^6	10^7	10^8	10^9
$\lceil \log_2 n \rceil$	17	20	24	27	30
$\lceil \log_{10} n \rceil$	5	6	7	8	9
$\lceil \log_{100} n \rceil$	3	3	4	4	5
$\lceil \log_{1000} n \rceil$	2	2	3	3	3

Multiway Search Tree of Order $m = 4$



- In an m-ary search tree, at most $m-1$ keys are used to decide which branch to take
- The data records are associated only with leaves, so the worst-case and average-case searches involve the tree height and the average leaf depth, respectively

B-Tree Definition

- A B-tree of order m is defined as an m-ary balanced tree with the following properties:
  - The root is either a leaf or it has between 2 and m children inclusive
  - Every nonleaf node (except possibly the root) has between $\lceil m/2 \rceil$ and m children inclusive

B-Tree Definition

- A nonleaf node with μ children has $\mu-1$ keys ($\mathbf{key}_i$: i=1, …, $\mu-1$) to guide the search
- $\mathbf{key}_i$ represents the smallest key in the subtree i+1
- All leaves are at the same depth
- The data items are stored at leaves, and every leaf contains between $\lceil l/2 \rceil$ and l data items, for some l that may be chosen independently of the tree order m
- Assuming that each node represents a disk block, the choice is based on the size of items that are being stored

Example of choosing m, l

- Let one data block hold 8192 bytes and each **key** use 32 bytes
- Let there be 10^7 data records, 256 bytes per record
- B-tree of order m: $m-1$ keys per node plus m branches
- The branch is a number of another disk block, so let a branch be 4 bytes:

  $32(m-1)+4m = 36m-32 < 8192 \rightarrow m=228$
- Because the block size is 8192 bytes, $l = 32$ records of size 256 bytes fit in a single block

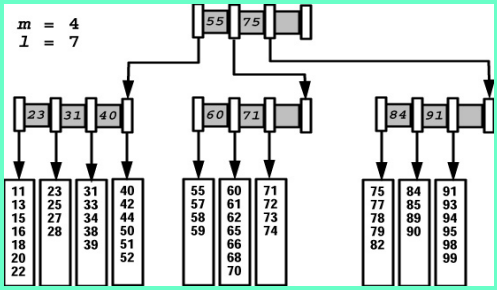
Example of choosing m, l

- Each leaf has between 16..32 data records inclusive, and each internal node, except from the root, branches in 114 – 228 ways
- 10^7 records can be stored in 312500 – 625000 leaves (= 10^7 / (16 .. 32))
- In the worst case the leaves would be on the level 4 ($114^2 = 12996 < 625{,}000 < 114^3 = 1481544$)

Naming the B-Trees

- B-trees are named after their branching factors, that is, $\lceil m/2 \rceil - m$
- A 4–7-tree is the B-tree of order $m = 7$
  - 2..4 children per root
  - 4..7 children per each non-root node
- A 3–4-tree is the B-tree of order $m = 4$
  - 2..3 children per root
  - 3..4 children per each non-root node

3–4-Tree with 4–7 items per leaf



Analysis of B-Trees

- A search or an insertion in a B-tree of order m with n data items requires fewer than $\lceil \log_m n \rceil$ disk accesses
- In practice, $\lceil \log_m n \rceil$ is almost constant as long as m is not small
- Data insertion is simple until the corresponding leaf is not already full; then it must be split into 2 leaves, and the parent(s) should be updated

Analysis of B-Trees

- Additional disk writes for data insertion and deletion are extremely rare
- An algorithm analysis beyond the scope of this course shows that both insertions, deletions, and retrievals of data have only $\log_{m/2} n$ disk accesses in the worst case (e.g., $\lceil \log_{114} 625000 \rceil = 3$ in the above example)