B-Trees

- Algorithms and data structures for external memory as opposed to the main memory
- B-Trees

Previous Lecture

- Height balanced binary search trees: red-black trees, AVL trees
- Advantages: search faster than in an average binary search tree because the tree is guaranteed to have the smallest possible depth.
- Today: non-binary trees (B-trees)

Two types of memory

- Main memory (RAM)
- External (peripheral) storage: hard disk, CD-ROM, tape, etc.
- Different considerations are important in designing algorithms and data structures for primary (main) versus secondary (peripheral) memory.

External storage

- So far, we analysed data structures assuming that all data is kept in main memory.
- If the data is kept in external memory, should take into account disk access time.
- It takes much longer to find the right place on disk compared to main memory.

Main principles

Data is stored on disk in chunks (pages, blocks, allocation units) and the disk drive reads or writes a minimum of one page at a time. To optimise an algorithm for accessing external memory, should

- minimise disk accesses
- read and write in multiples of page sizes.

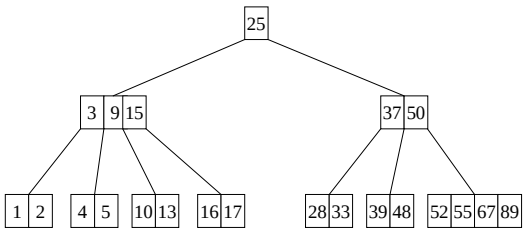
B-trees

- A good example of a data structure for external memory is a B-tree.
- Better than binary search trees if data is stored in external memory (they are NOT better with in-memory data!).
- Proposed by R. Bayer and E. M. McCreigh in 1972.

B-trees

- Each node in a tree should correspond to a block of data.
- Each node stores many data items and has many successors (stores addresses of successor blocks).
- The tree has fewer levels but search for an item involves more comparisons at each level.
- For database search, it is more important to minimise page swaps than comparisons.

Example of a B-Tree



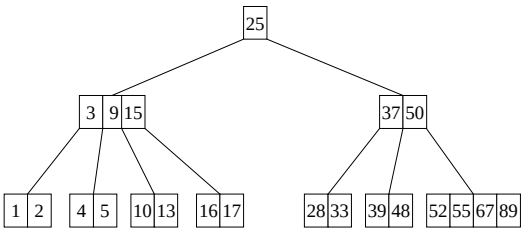
Definition of a B-Tree of order m

- There is a single root node, which may have as few as two daughters (or none if the tree is empty).
- All other non-leaf nodes must have between m/2 and m daughters.
- Each node stores at most m-1 ordered items.
- All items in the *n*th subtree of a node are less than the *n*th item (if exists) and greater than the *n-1* item (if exists).
- All leaves on the tree must be at the same level.

Practical considerations

- Since each node correspond to a block/page, their size is usually a power of 2.
- The number of records in a node is therefore usually even.
- The number of successors (the order of the tree) is therefore usually odd.

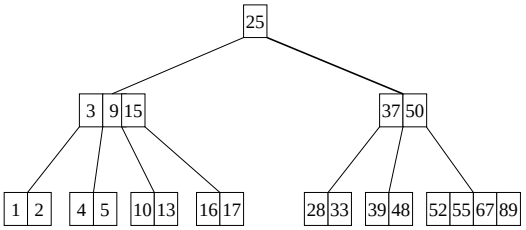
B-Tree of order 5:



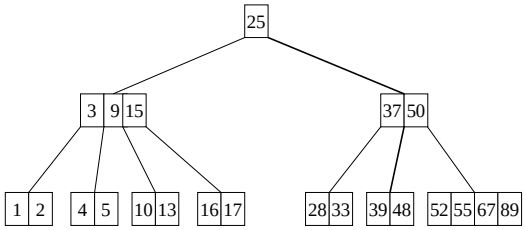
Search in a B-Tree

- The search for a record with key x is similar to the binary tree search.
- Search the root for x. Either x is found, or x falls in a place between two keys (or before the first one, or after the last one) and the corresponding subtree should be searched.

Search for record with key 40:



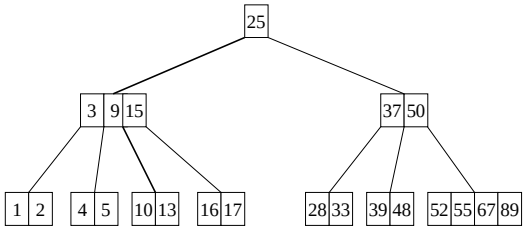
Search for record with key 40:



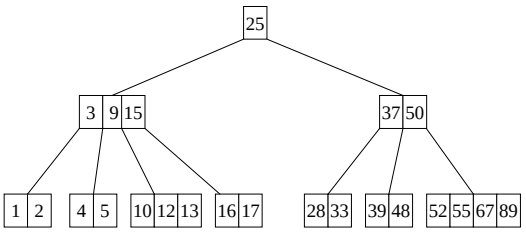
Inserting in a B-Tree

- Find the node where the item is to be inserted by following the search procedure.
- If the node is not full, insert the item into the node in order.
- If the node is full, it has to be split.

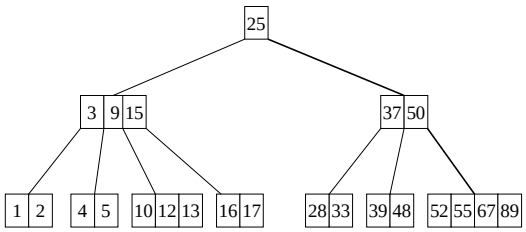
Inserting 12:



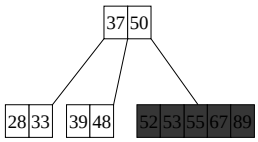
Inserting 12:



Inserting 53:



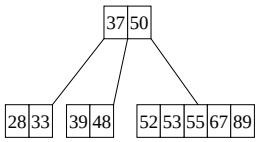
Inserting 53: the node is too full!



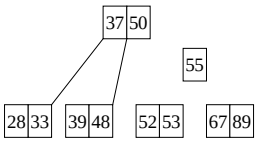
Splitting a node

- Find middle value (of old items in the node and the new item).
- Keep items smaller than middle in the old node.
- Put items greater than middle in the new node.
- Push middle item up into parent node.

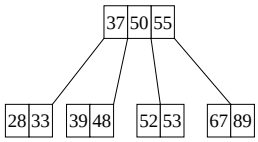
Splitting a node



Splitting a node



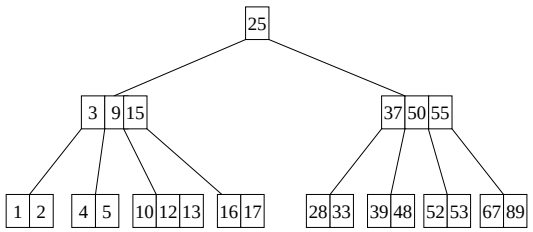
Splitting a node



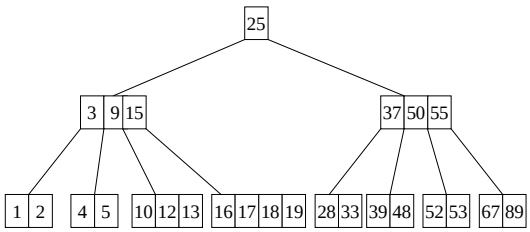
Insertion

- If this makes the parent full, split it as well and push its middle item upwards.
- Continue doing this until either some space is found in an ancestor node, or a new root node is created.

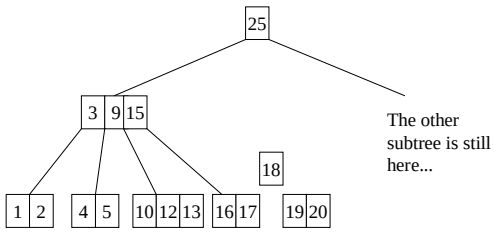
Please insert 18,19,20,21,22,23:



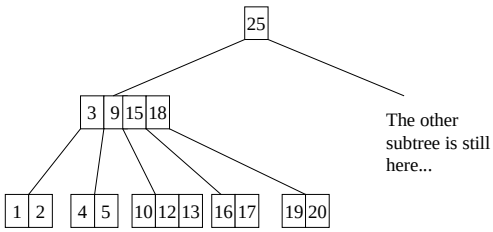
Insert 18,19



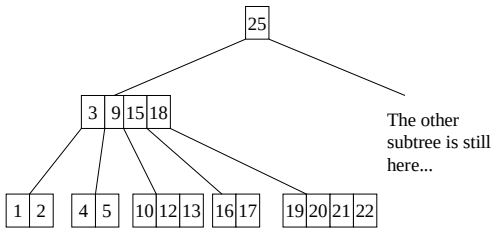
Insert 20



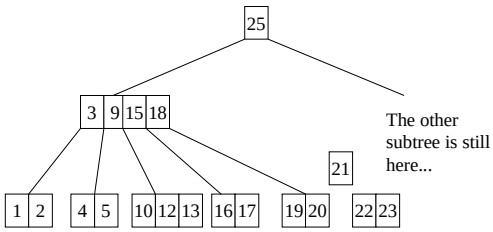
Insert 20



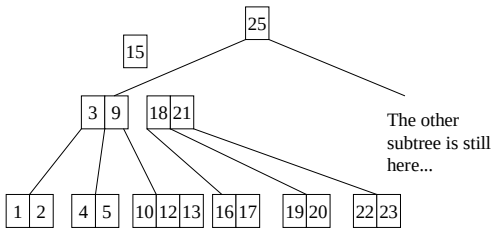
Insert 21,22



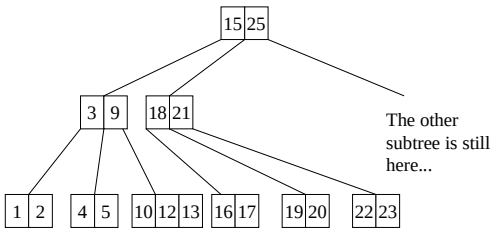
Insert 23



Insert 23



Insert 23



Efficiency of B-trees

- If a B-tree has degree M, then each node (apart from the root) has at least M/2 (round up for odd M) successors. So the depth of the tree is at most $log_{M/2}(size)+1$. In the worst case have to make M-1 comparisons in each node.
- Fewer disk accesses than for a binary tree.
- Each node could correspond to one block of data (plus addresses of successor nodes).

Variant of B-trees

- Only leafs contain data
- Non-leafs contain only keys and block numbers
- Access speed is higher because each block can hold more block numbers.
- Programming is more complicated because there are two kinds of nodes: leafs and non-leafs.

Indexing

- Index: list of key/block pairs.
- If small enough, could be kept in the main memory.
- If the index is too large to be loaded in the main memory, it is often kept in a B-tree.
- Multiple index files: if different kinds of search have to be performed, can keep one index for every set of keys.

Summary and reading

- Very important: the issues involved in designing algorithms and data structure for use with external memory are different!
- Compulsory reading: Shaffer 9.1, 9.2 (external memory), 11.5 (B-trees) or Lafore pp.400-410 (Chapter 10).
- External sorting (optional)
- Indexing