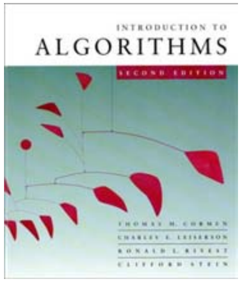
CS 5633 -- Spring 2005



B-trees

External memory dictionary

Task: Given a large amount of data that does not fit into main memory, process it into a dictionary data structure

- Need to minimize number of disk accesses
- With each disk read, read a whole block of data
- Construct a balanced search tree that uses one disk block per tree node
- Each node needs to contain more than one key

k-ary search trees

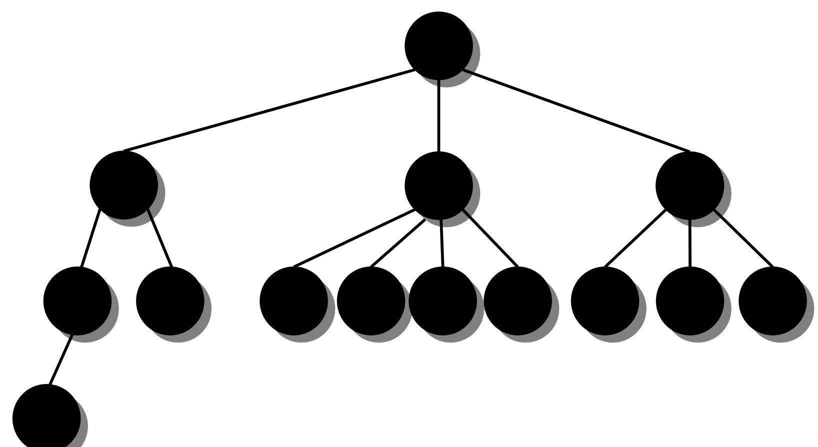
A ***k*-ary search tree** T is defined as follows:

•For each node x of T:

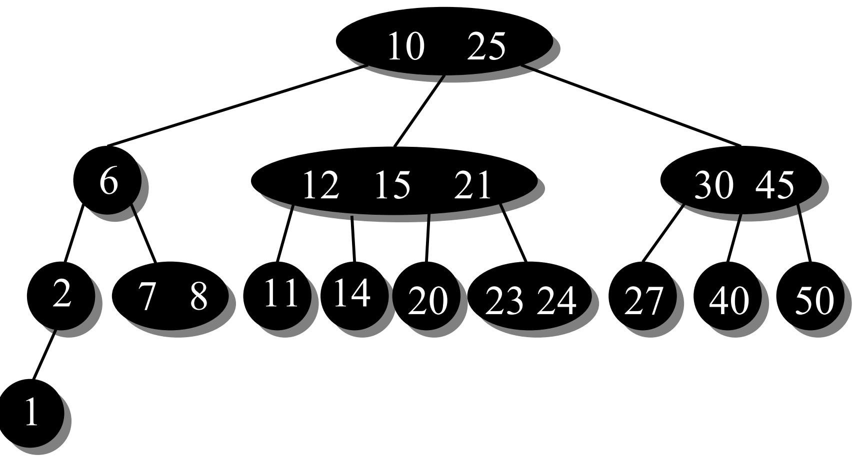
- x has at most k children (i.e., T is a k-ary tree)
- x stores an ordered list of pointers to its children
- x stores an ordered list of keys ($1 \leq$ # keys $\leq k-1$, and # keys $\geq$ # children – 1)
- x fulfills the search tree property:

keys in subtree rooted at i-th child $\leq$ i-th key $\leq$ keys in subtree rooted at $(i+1)$-st child

Example of a 4-ary tree



Example of a 4-ary search tree

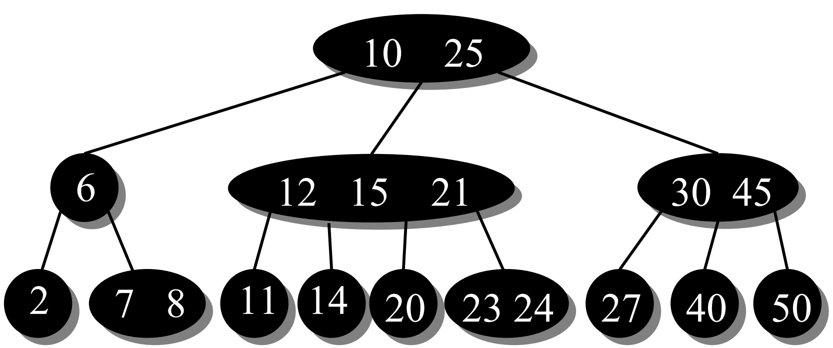


B-tree

A ***B*-tree** T with **minimum degree** $k \geq 2$ is defined as follows:

- T is a $(2k)$-ary search tree
- For every internal node: #keys = #children-1
- Every node, except the root, stores at least k-1 keys (every internal non-root node has at least k children)
- The root must store at least one key
- All leaves have the same depth

B-tree with k=2



Remark: This is a (2,3,4)-tree.

Height of a B-tree

Theorem: A B-tree with minimum degree $k \geq 2$ which stores n keys has height h at most

$$\log_k (n+1)/2$$

Proof: #nodes $\geq 1+2+2k+2k^2+\ldots+2k^{h-1}$

level 1 level 3

level 0 level 2

$$n = \#\text{keys} \geq 1+(k-1)\sum_{i=0}^{h-1} 2k^i = 1+2(k-1)\cdot\frac{k^h-1}{k-1} = 2k^h-1$$

B-tree search

B-TREE-SEARCH(x,key)

$i \leftarrow 1$

while $i \leq \#keys\ of\ x$ **and** $key > i$-th key of x

do $i \leftarrow i+1$

if $i \leq \#keys\ of\ x$ **and** $key = i$-th key of x

then return (x,i)

if x is a leaf

then return NIL

else b=DISK-READ(i-th child of x)

return B-TREE-SEARCH(b,key)

B-tree search runtime

- $O(k)$ per node
- Path has height $h = O(\log_k n)$
- CPU-time: $O(k \log_k n)$

- Disk accesses: $O(\log_k n)$

  disk accesses are more expensive than CPU time