B-trees

- very important data structure in computer science
- database indexing, hard disk referencing, MongoDB, ...
- balanced, multi-way search tree
- many slight variations, we will use definition in CLRS text
- idea is that nodes are large and fit into a disk block (minimum amount of data that’s pulled off a hard drive)
- node size parameters (here called t) depend on disk speeds, block sizes, etc.
B-tree specifications

- fixed parameter $t$, called *minimum degree*
- nodes have between $t-1$ and $2t-1$ keys
- so therefore they have between $t$ and $2t$ children
- root is exception: it may have as few as 1 key (2 children)
- all null pointers have the same depth (distance from root)
- a 2-3-4 tree is a B-tree with minimum degree $t=2$
B-tree node format

• each node can have between $t$ and $2t$ children
• a node might look like
  • $<P_0, K_1, P_1, K_2, P_2, ... , P_{q-2}, K_{q-1}, P_{q-1}>$
  • for $t \leq q \leq 2t$ (except for root $2 \leq q \leq 2t$)
• during a search, we split full nodes
• a node is full when it has $2t-1$ keys
node split (shown for $t=3$)

split a full node

becomes

promote $c$ by inserting into parent
B-tree height

• theorem 18.1: if \( n \geq 1 \), then for any B-tree containing \( n \) keys of height \( h \) and minimum degree \( t \geq 2 \), \( h \leq \log_t \frac{n+1}{2} \).

• example: \( t=50 \) and \( n=100,000,000 \)
  • \( h \leq \log_{50} 50,000,000.5 \approx 4.53 \leq 5 \)

• suppose 20 records fit on a page

• without the index to find an item we’d need to search about half the \( 100,000,000/20=5,000,000 \), resulting in 2,500,000 disk accesses

• with the index we need at most 5+1=6 disk accesses (5 for the tree nodes and one for the page containing that key’s record)
exercise 18.2-1

Insert into initially empty B-tree of min degree t=2 the key values F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E

After the first 3 values:

\(<F, Q, S>\)

A search to place K causes a split:

After which K is placed:
place C

search for L splits full node:

split

insert
insert M
ETC....
aside: splay trees

- maintain balance kind of
- insertion, deletion, union take $O(lg n)$ amortized time
  - a series of $m$ of these operations on $n$ keys takes total time worst case $O(m*lg n)$
  - possible that one operation takes $O(n)$ time but cannot happen often
- NO balance information needs to be stored at node (balance factor, color)
- used in DNS servers sometimes
splay trees use rotations

• idea is that whenever a node is accessed, it is moved to the root by a series of rotations
  • LOTS OF ROTATIONS!
  • and slightly different ones
• if x is child of root, it is moved upwards with a single rotation
  • called a ZIG rotation
• if x has a (RL or LR) grandparent, it is moved up with a double rotation
  • called a ZIG-ZAG rotation
• if x has a (LL or RR) rotation, then moved up with a special rotation
  • ZIG-ZIG
• idea: zig-zigs and zig-zags tend towards rebalancing
zig-zig
zig-zag
example: find 7
zig-zig,

x, and done