Logistics

• Milestone #2 of PA02
• Review red-black tree
• The relation between red-black tree and 2-3-4 tree
• B-tree
Insert 14, 15, delete root
2-3-4 tree

- Nodes store 1, 2, or 3 keys and have 2, 3, or 4 children, respectively.
- All leaves have the same depth.
2-3-4 tree

- 2-node
  - 1 key, 2 kids
  - Same a binary tree

- 3-node
  - 2 keys and 3 kids

- 4-node
  - 3 keys and 4 kids
Insert to 2-3-4 tree

• Insert the **new key** at the **lowest internal node** reached in the search
  • 2-node becomes 3-node

![Diagram of 2-node becoming 3-node](image)

• 3-node becomes 4-node

![Diagram of 3-node becoming 4-node](image)

• What about a 4-node?
Insert to 2-3-4 tree

• In our way down the tree, whenever we reach a 4-node, we break it up into two 2-nodes, and move the middle element up into the parent node.
2-3-4 tree and Red-black tree

- 2-3-4 tree and red-black tree are closely related
B-tree

• “B” means several things: balanced, Boeing,

• Motivated by
  • Limited resources in early years
  • Data that cannot fit into the main memory at once
  • Even today, we have gigabytes of memories, we still face similar problem
    • Better data acquisition techniques (many more sensors)
    • More data capturing devices (your cellphone)
    • More people are connected
    • Mobile devices also have limited resources
    • Resources can be shared by many people (cloud computing/storage)
Problem

• Secondary storage devices is 1 million times (at least) slower than the primary storage and even more so than L1 and L2 cache!
  • Reading each node that stores on the disk might take 1/10 second
  • Even for a red-black tree it will take $O(\log_2 n)$ reads from the disk
  • For 10 million record, the height of a red-black tree is about 25
  • That is 25 disk accesses

• Solution?
Problem

• Data (record) size might not be uniform (some can be much bigger)
  • Store index (key or address)

• Properties
  • Disk seek time is very slow, but reading data (once it is found on disk) is much faster
  • So, we should read a much as possible once a record is found
  • Usually a block of data will be read from the disk anyway, even if you just ask for a byte
  • This implies that, each node should store entire disk block (page size)
  • Each node should also store index of the record instead of the data itself
  • So, we are talking about something like 512-ary tree

• Finally, how do we balance, insert, delete data from such a tree?
B-tree

• Data items are all in the leaves
• Root is a leaf or has 2 to $M$ children
• Non-leaf node has $M - 1$ indices (keys)
  • Key $i$ is the smallest key in $(i + 1)$–th subtree
  • Must have $\left\lceil \frac{M}{2} \right\rceil$ to $M$ children
• All leaves are at the same level and have $\left\lceil \frac{L}{2} \right\rceil$ to $L$ data
• Note $M$ and $L$ are user inputs
B-tree

• Insert($x$)
  • Find the leaf that $x$ correspond to
  • Adjust the tree so it does not violate the B-tree properties
    • Split a node into two nodes
    • Promote the index (key) to a parent node

• Delete($x$)
  • Find the leaf that $x$ correspond to
  • Adjust the tree so it does not violate the B-tree properties
A B-tree of order 5
Insert 57
• After inserting 57
• Insert 55
• After inserting 55
• Insert 40
• After inserting 40
• Delete 99
• After deleting 99