Storage and Retrieval

Data at Scale
Types of Data Structures

- Organize data to make access / processing fast
  - Speed depends on the internal organization
  - Internal organization allows different types of accesses

- Problems:
  - Large data is nowadays distributed over several data centers
  - Need to take advantage of storage devices
Types of Data Structure

• Dictionary — Key - Value Store
  • CRUD operations: create, read, update, delete
  • Solutions differ regarding read and write speeds
Types of Data Structure

- Range Queries (Big Table, RP)
- CRUD and range operation
Types of Data Structure

- Priority queue:
  - Insert, retrieve minimum and delete it
Types of Data Structure

• Log:
  • Append, Read
B-Trees

• B-trees: In memory data structure for CRUD and range queries
  • Balanced Tree
  • Each node can have between $d$ and $2d$ keys with the exception of the root
  • Each node consists of a sequence of node pointer, key, node pointer, key, ..., key, node pointer
  • Tree is ordered.
    • All keys in a child are between the keys adjacent to the node pointer
B-Trees

- Example: 2-3 tree: Each node has two or three children
B-Trees

• Read dog:
  • Load root, determine location of dog in relation to the keys
  • Follow middle pointer
  • Follow pointer to the left
  • Find “dog”
B-Trees
B-Trees

• Search for “auk”:

- ant
- asp
- bug
- cow
- cur
- doe
- dog
- elk
- emu
- kid
- pen
- rat
- zho
- dab
- kea
- olm
- ram
B-Trees

- Range Query \(c - l\)
- Determine location of \(c\) and \(l\)
B-Trees

• Recursively enumerate all nodes between the lines starting with root
B-trees

- Capacity: With \( l \) levels, minimum of \( 1 + 2 + 2^2 + \ldots + 2^l \) nodes:
  - \( 1(2^{l+1} - 1) \) keys
- Maximum of \( 1 + 3 + 3^2 + \ldots + 3^l \) nodes:
  - \( \frac{2}{2}(3^{l+1} - 1) \) keys
B-trees

- Inserts:
  - Determine where the key should be located in a leaf
  - Insert into leaf node
  - Leaf node can now have too many nodes
  - Take middle node and elevate it to the next higher level
  - Which can cause more “splits”
B-trees
B-trees
B-trees
B-trees

- Insert: Lock all nodes from root on down so that only one process can operate on the nodes
- Tree only grows a new level by splitting the root
B-Trees

- Using only splits leads to skinny trees
  - Better to make use of potential room in adjacent nodes
  - Insert “ewe”.
    - Node elk-emu only has one true neighbor.
      - Node kid does not count, it is a cousin, not a sibling
B-tree

- Insert ewe into
B-tree

- Insert ewe
• Promote elk. elk is guaranteed to come right after eft.
• Demote eft
B-tree

- Insert eft into the leaf node
B-tree

- Left rotate
  - Overflowing node has a sibling to the left with space
- Move left-most key up
- Lower left-most key
B-tree

Now insert “ai”
B-tree

Insert creates an overflowing node
Only one neighboring sibling, but that one is full
Split!
Middle key moves up
Unfortunately, this gives another overflow
But this node has a right sibling not at full capacity
B-tree

Right rotate:
Move “bot” up
Move “doe” down
Reattach nodes
B-tree

Move “bot” up
Move “doe” down
Reattach the dangling node
“bot” had moved up and replaced doe

The “emu” node needs to receive one key and one pointer
B-tree
B-tree

• Deletes
  • Usually restructuring not done because there is no need
  • Underflowing nodes will fill up with new inserts
B-tree

• Implementing deletion anyway:
  • Can only remove keys from leaves
  • If a delete causes an underflow, try a rotate into the underflowing node
  • If this is not possible, then merge with a sibling
    • A merge is the opposite of a split
  • This can create an underflow in the parent node
    • Again, first try rotate, then do a merge
Delete “kit”
“kit” is in an interior node.
Exchange it with the key in the leaf immediately before “fox”
After interchanging “fox” and “kit”, can delete “kit”
Now delete “fox”
Step 1: Find the key. If it is not in a leaf
Step 2: Determine the key just before it, necessarily in a leaf
Step 3: Interchange the two keys
Step 4: Remove the key now from a leaf
This causes an underflow
Remedy the underflow by right rotating from the sibling
B-tree

Everything is now in order
B-tree

Now delete fly
Switch “fly” with “emu”
remove “fly” from the leaf
Again: underflow
Cannot left-rotate: There is no left sibling
Cannot right-rotate: The right sibling has only one key
Need to merge: Combine the two nodes by bringing down “elk”
We can merge the two nodes because the number of keys combined is less than $2k$
Delete “emu”
Switch predecessor, then delete from node
B-tree

Now delete “elk”
Results in an underflow
B-tree

Results in an underflow
But can rotate a key into the underflowing node
B-tree

Result after left-rotation
B-tree

“Now delete “eel”
B-tree

Interchange “eel” with its predecessor
Delete “eel” from leaf:
Underflow
B-tree

Need to merge
Merge results in another underflow
Use right rotate
(though merge with right sibling is possible)
B-tree

“ass” goes up, “bot” goes down
One node is reattached
B-tree

Reattach node
B-tree

ant

ai

ape

auk bat

bug cat

ox

koi owl

rat sow

ass doe
In real life

- Use B+ tree for better access with block storage
  - Data pointers / data are only in the leaf nodes
  - Interior nodes only have keys as signals
- Link leaf nodes for faster range queries.
B+ Tree
B+ Tree

• Real life B+ trees:
  • Interior nodes have many more keys (e.g. 100)
  • Leaf nodes have as much data as they can keep
  • Need few levels:
    • Fast lookup