Case study

Problem: Find a way to store student records for a course, with unique IDs for each student, where records can be accessed, added, or deleted.
ADT Dictionary

Data: (key, element pairs) where

- keys are distinct but not necessarily orderable, and
- elements are any data.

Operations:

- CreateDict()
- IsEmptyDict(D)
- LookUpDict(D, key)
- AddToDict(D, key, element)
- DeleteFromDict(D, key)

Note: AddToDict and DeleteFromDict both require a use of LookUpDict.
Optional operation for restriction with orderable keys:

- ClosestKeyBefore(k)
Additional operations needed to store values

- KeyAtNode(B, node)
- ElementAtNode(B, node)
- StoreInNode(B, node, key, value)
Implementing \text{DeleteFromDict}(D, \text{key})

Find the node \( n \) containing the key

Case 1: \( n \) has no children: delete \( n \)

Case 2: \( n \) has only one child \( c \): delete \( n \), make \( c \) into the child of \( n \)’s parent

Case 3: \( n \) has two children:

- Find the inorder successor \( s \) of \( n \).
- Replace \( n \)’s key with \( s \)’s key.
- Delete the node containing \( n \).
Binary search tree deletion, illustrated

Case 1: 12, 29, 35, 50
Case 2: 10, 15, 27
Case 3: 20, then 27 (Case 2)
Case 3: 30, then 35 (Case 1)
Case 3: 40, then 50 (Case 1)
AVL trees

- A node is **balanced** if the difference in height of left and right subtrees is at most 1.
- A tree satisfies the **height-balance property** if every node is balanced.
- An **AVL tree** is a height-balanced BST.
AVL tree example

Balance: 0 if even, 1 if left higher, -1 if right higher

```
  20
 /  \
10   30
 /  +1  \
0   27
 /     |
0  22 0
/    |
0  35 0
```

```
-1
```

0
0
0
0
0
The **pivot node** is the lowest unbalanced node in the tree after a new node has been inserted.

A **rotation on the pivot node** is a rearrangement of subtrees that rebalances the tree without violating binary search order.

**Algorithm:**

- Find leaf to insert as in BST.
- Trace path from leaf to root to identify the pivot node.
- Rebalance by executing a rotation on the pivot node.
• Insertion in A causes imbalance at D.
• Letters show BST order preserved.
• Balance at B means A is $h+1$ and C is $h$.
• Imbalance at D means E is $h$.
AVL AddToDict case 2

- Insertion at C or E causes imbalance at F.
- Balance at B means A is h and one of C and E is h, the other h-1.
- Imbalance at F means G is h.
AVL AddToDict case 3

- Insertion at E causes imbalance at B.
- Balance at D means C is h.
- Imbalance at B means A is h.
• Insertion at C or E causes imbalance at B.
• Balance at D means C and E both h-1 or h.
• Balance at F means G is h.
• Imbalance at B means A is h.
AVL tree DeleteFromDict

Algorithm:
- Delete as in BST.
- Trace up to find first unbalanced node $z$.
- $y =$ child of $z$ with greater height
- $x =$ child of $y$ with smaller height (maybe both same)
- Rebalance using rotation
- ***Keep tracing up to see if another unbalanced node higher.***
Multiway search trees

Properties:

- Ordered tree.
- Each internal node has at least 2 children.
- A d-node (with d children) has d-1 (key, element) pairs, with keys in order $k_1, \ldots, k_{d-1}$.
- Any item $(k,e)$ in the subtree of $v$ rooted at the $i$th child of $v$ has $k_{i-1} \leq k \leq k_i$ (view $k_0 = -\infty$ and $k_d = \infty$).
(2,3) tree

(2,3) tree definition

- Each internal node has either one key and two children or two keys and three children.
- All leaves are at the same depth and have one or two keys.
If there are too many values in a node, then **overflow** has occurred.

The **split** operation splits a node into two and rearranges values as needed.

Basic idea of operation:

- Search leads to a leaf.
- Add item to leaf.
- If overflow, split leaf.
- If splitting the leaf leads to overflow in the parent, then the parent may need to be split as well.
- Splitting can propagate up to the root.
Basic idea of split

- Node with three keys becomes two nodes, one with smallest and one with largest.
- Middle key is sent up to the parent.
- If the node being split is the parent, the tree now has a new root and is one taller.
Add 22.

(2,3) tree AddToDict single split
Add 57.

(2,3) tree AddToDict cascading split

Add 57.
If there are too few values in a node, then **underflow** has occurred.

The **fuse** operation fuses two nodes into one and rearranges values as needed.

Basic idea of operation:

- Search leads to a leaf or internal node.
- If internal node, like in BST swap with inorder successor.
- Remove item from leaf.
- If underflow, need to fuse nodes.
- If fusing leads to underflow in the parent, then the parent and its sibling may need to be fused as well.
- Fusing can propagate up to the root.
Definitions concerning memory

The **memory hierarchy** consists of different type of memory, where the size of each type increases as the access speed decreases:

- registers
- cache (multiple levels)
- main memory
- secondary storage
- tertiary storage

A **page** is a fixed-size chunk of data that is moved between levels.

A **paging algorithm** is used to determine which page(s) to move out when a new page is moved in.
B-tree

B-tree of order $d$:

- $d$ chosen so that $d$ data items fill a page.
- Root has at most $d$ children.
- Other nodes have at least $\lceil d/2 \rceil$ and at most $d$ children.
- Height is $O(\log \lceil d/2 \rceil n)$ and $\Omega(\log_d n)$.

Operations:

- $d + 1$ children split into nodes with $\lceil (d + 1)/2 \rceil$ and $\lfloor (d + 1)/2 \rfloor$ items.
- $\lceil d/2 \rceil - 1$ children joined to become a node of size at most $d$.

Space usage as low as 50%, average 69%.