AVL Deletion (key by node)
- Delete node as per BST deletion
- Moving upwards from point of deletion, update height as required and find first point of imbalance if any.
- Use trimode restructuring to rebalance an unbalanced node
Multiway search trees

Def: A multiway search tree is one with full property

- Each node has at least 2 children

- Each node contains a collection \( k_1, \ldots, k_s \) of keys

- A node with \( s \) keys has \( s + 1 \) subtrees associated with it.

- All keys in subtree \( i \) are at:

  \[ k_i \leq x < k_{i+1} \]

  where \( k_0 = -\infty \) and \( k_{s+1} = +\infty \)

Find 33
Find in multiway search tree

Root is \( k_1, \ldots, k_i \), find \( x \)

\[
\text{Find}(x, \text{Dict}(k_1, \ldots, k_i))
\]

\( \checkmark \) Yes

\( \nabla \) No \((x \not\in k_i) \rightarrow k_i, k_{i+1} \)

Recursively continue search in subtree lying to the left of \( k_i \), and

\( k_{i+1} \) in subtree \( i+1 \).
Multinary search trees restart

Def: We call a (general) tree node a $d$-node if it has $d$ children.

Def: A multinary search tree node is a $d$-node ($d \geq 2$) that contains $d-1$ keys $k_1 \leq k_2 \leq \ldots \leq k_d$.

Def: A multinary search tree is a tree with the property that
- All leaves are $d$-nodes $(k_1, \ldots, k_{d-1})$.
- If an internal node $v$ is a $d$-node then it has $d$ subtrees $T_{v,1}, \ldots, T_{v,d}$ s.t.:

\[
\min \text{ key}(T_{v,1}) \leq k_1 < \min \text{ key}(T_{v,2}) \leq \min \text{ key}(T_{v,3}) \leq k_2 \ldots
\]
Find in a multivary search tree \( x \)

- Start from root

- At an internal node \( v = (k_1, \ldots, k_{d-1}) \) run a Dictionary-Search \( \& \) on \( k_1, \ldots, k_{d-1} \).

  - If Dictionary-Search \( \& \) returns found then return with key

  

  \[
  \begin{align*}
  \text{else} & \quad \text{D-search returns } \ i \ \text{of } 1 \leq i \leq d \\
  \text{such that if } x \text{ exists it must lie in } T_v, i.
  \end{align*}
  \]

  Then continue with Search \( (T_v, i, x) \).

D-Search returns \( x_-, x_+ \).

If \( x = x_+ \) then \( i = 1 \), if \( x = x_- \) then \( i = 2 \), else \( i \) in the index of \( x_+ \).
2-4 Trees

Definition: A 2-4 tree is a multiway search with keys restricted to 0, 2, 3, 8. Additionally, all leaves have the same depth.

Q: Given n keys what is the height h of a 2-4 tree.

If no of nodes at depth d = $n_d$

$2^{n_0} \leq n_{d+1} \leq 4n_d$

Total no of nodes = $\sum n_i$

$2^{n_0} \leq n_{d+1} \leq 4n_d$

$4^{h+1} - 1$ keys
\[ \Rightarrow \quad \# \text{keys} \leq 4^{h+1} - 1 \]

and
\[ \# \text{keys} \geq 2^{h+1} - 1 \]

\[ \log_4(n+1) \leq h \leq \log_2(n+1) \]
Alg: Insert (x)
- Search (x)
  - to reach a leaf
  - Insert x in the parent of the leaf
  - If no overflow
    - stop
  - If overflow
    - promote 2nd or 3rd key to parent
    - and repeat from
  - If no parent
    - create new root.
2-A tree deletion

Alg:
- if key is not in
  lowest level replace
  it with appropriate
  key from lowest level ("move
  predecessor/successor")
and do leaf-level deletion
- leaf-level deletion
  - delete key if in underflow stop,
  else
Generalization:

- \((x, y)\) trees.

- S+ Trees (self-study)

Red-black trees \(\rightarrow\) Balanced BST.

**Sorting:**

\[ D = \{ a, b, \ldots \} \]

\[ S : D \rightarrow \{ x_1, \ldots, x_n \} \]

\[ \text{if } i < j \Rightarrow x_i \in x_j \]
Sorting is the process of finding an order respecting permutation of the input sequence \( \Rightarrow \) \( 1 \) (in the w.c.)

sequence has to be found out of \( m! \) \( \Rightarrow \) \( \Theta (m \log m) \)
time in the worst case.

Algorithms already seen
- Insertion sort \( \Rightarrow \Theta (m^2) \) time
- Selection sort \( \Rightarrow \Theta (m^2) \) time
- Heap sort \( \Rightarrow \Theta (m \log m) \) time
- There are n rounds
- In each round, start from A[0]
  and move through the array
  comparing A[i] to A[i+1] for i = 0 to n-2
  If they are in the wrong order, flip them.

Θ(n^2) \( \leq \) Time

Round i ensures i\text{th} largest reaches its correct position.
Merge sort
- If array size = 1 return array
- else
  - Divide array into 2 equal parts
  - Recursively merge sort each part
  - Merge the two sorted arrays.
Merge sort takes $\Theta(n \log n)$ time.

Quick sort

- Pick an element $x$ (arbitrarily) from $A[\cdot]$.
- Create two arrays $B[\cdot]$ and $C[\cdot]$ so that $B[\cdot]$ contains elements $\leq x$ and $C[\cdot]$ contains elements $> x$.
- Recursively quicksort $B[\cdot]$ and $C[\cdot]$ and output as $B[\cdot] \cup C[\cdot]$. 
\[ T(n) = T(k_1) + T(k_2) \]

where \( k_1 + k_2 = n - 1 \)

\[ T(n) = T(n-1) + n \]

\[ T(1) = 1 \]

\[ q_5(A) \quad q_5(A, 0, n-1) \quad q_5(A, 0, 8) \]

Pivoting

\[ q_5(A, 0, 3) \quad q_5(A, 5, 8) \]
Pivoting in-place:

- \( A[0] \) is called the pivot

The process of recasting the array into two parts is called pivoting.

\[ \text{\[\leq a \}\[> a\]} \]

- Proceed with front and rear pointers (initialize to 0 and \( n-1 \) resp.)

\[ \begin{align*}
\text{hyp.:} & \quad A[0] \ldots A[f-1] \leq a \\
& \quad A[r+1] \ldots A[n-1] > a \\
\end{align*} \]

- Proceed to separate \( \{ y < a \} \cup \{ y > a \} \)

- At then end put \( A[0] \) at the last location of the left array (\( y < a \))
$\text{Running time}$

$0$

$k_0 \leq \frac{2}{3} k_0 \Rightarrow k_2 \geq \frac{2}{3} k_1$

If $\text{good} = \text{middle} \frac{1}{3}$

$k_i \leq M \left( \frac{2}{3} \right)^i$

$i = \log_{\frac{2}{3}} n$

\[ \Rightarrow \left( \frac{2}{3} \right)^i = \frac{1}{n} \]
good parent - child #a relationship
  = child #a gets ≤ 2/3 of parents key.

good parent = (good #a relationship) & (good #b relationship)

**bottomline**: Since in the real world keys are often randomly distributed Q5 works well in practice b/c if they are randomly distr then
r. t. = Θ(n log n)
Quicksort 2.0?

- Instead of choosing A[i] as pivot, spend Θ(n) time to find the median of A[n].
- Rest remains the same.

Randomized Quick Select

Q: Given A[1] find kth smallest element. (Θ(k+nlogn))

- Pick A[0] = x.
- Pivot A[0] around x & say minimum A[j].
  - If j > k: recurse (A, 0, j, k)
  - If j < k: recurse (A, j, n-1, k-j)
Bucket Sort

Suppose we have n keys all belonging to \( \{0, \ldots, N-1\} \)

1. Go through \( A \) from 0 to \( n-1 \)
2. If \( A[i] = k \) then insert \( i \) in \( B[k] \)'s list.
3. Sort the lists in order \( B[0] \ldots B[N-1] \)

\[
\text{Running time} = \Theta(n+N)
\]
Radix sort:

Def: lexicographical ordering of “strings”

Suppose we have a totally ordered “alphabet” \( \{ a_1, a_2, \ldots, a_n \} \)

\( a = a_1 a_2 \ldots a_e \)

\( y = y_1 y_2 \ldots y_e \) are strings on our alphabet

We say that \( a \leq y \) if either \( a_i \leq y_i \) for all \( i \) or \( a_i = y_i \) for some \( i \) with \( a_i < y_i \)

\( \lambda = \text{empty string} \)

\( \lambda \preceq \text{non-empty string} \) and \( a_1 \ldots a_e \preceq y_1 \ldots y_e \)
Stable sorting

Given \((x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)\)

Sort on first coordinate to get

\((x_1', y_1'), (x_2', y_2') \ldots (x_n', y_n')\)

Suppose \(x_i, x_{i+1} \ldots x_j = x_k\)

The sort is said to be stable if the pairs

\((x_k, y_i), (x_k, y_{i+1}) \ldots (x_k, y_j)\)

appear in the same order in the sorted as in the unsorted sequence.
\[(7, 19), (6, 2), (7, 3), (5, 17), (7, 1), (8, 26)\]

\[
(5, 17), (5, 2), (7, 3), (7, 1), (7, 1), (8, 26)
\]

\[
(6, 2), (7, 19), (7, 3)
\]
273, 078, 932, 017, 645, 368

Given a digit numbers with columns $d, d-1 \ldots 1$.
From $i = 1$ to $d$

stably sort the current array by the $i$th column

$O(dn)$
Correctness by induction

IH: After i rounds the numbers are sorted if we only look the i least significant digits.

1. Sort column i+1. Now group the numbers by the elements of the alphabet. If a block the wins are sorted by IH & stability. Across blocks by correctness of your bucket sort.
The Graph ADT

**Defn:** A graph is a tuple \((V, E)\) where \(V\) is a set of "something" and \(E\) is a "sub" set of \(V \times V\).

**Twitter**
- \(V\) - set of users
- \(E\) - set of (follower, followed)

**Facebook**
- \(V\) - set of users
- \(E\) - set of friend pairs

**Academia**
- \(V\) - set of authors & researchers
- \(E\) - \((u, v) \in E \iff u \& v \text{ published together}\)
A Graph ADT method (V in ADT A)

Vertex operations - adjacent edges () - A x A List
adjacent vertices () - A List

Edge operations + Dynamic Operations (insert/delete etc)
Data structures

\[d = \text{max deg}(u)\]

Adjacency list

Adjacency matrix
6.11.2018

**Neo4J, Titan Graph DB**

Edge list

Graph traversal

Vertex based data structure

AdjList/Matrix
Breadth First Search (BFS, \( n \times O \))

- \( 0 \rightarrow \) 
- \( 1 \rightarrow \) 
- \( 2 \rightarrow \)

- Store newly discovered vertices in queue.
Learning time: