Today: Augmented Data Structures
  - Paradigm
  - Examples
    1. Dynamic Order Stacks
    2. Interval Search

Last lecture: 2-3 Trees
- Main property: Can add/delete keys, search/find successors in $O(\log n)$ time.
- Today: Can we do more with them?
Today:

- Will show how to use 2-3 trees to solve new problems.

- Idea: Add other information at nodes.
  - Show how to answer new queries with new info.
  - Show how to maintain new information during updates.

- "Augmented" Data Structures
**Review 2-3 Trees**

**Definition:**

1. All leaves at same depth;
2. All keys at leaves;
3. Internal nodes have 2 or 3 children.
4. Also store max key in each subtree.

**Search Property:**

\[
\max(\text{key}(\text{subtree}_{i-1})) < \min(\text{key}(\text{subtree}_i))
\]
Key Operations (during INSERT/DELETE)

- **ADD new child**:

- **SPlice child**: 
**SPLIT**

- $m_D \
  \downarrow \
  m_A \quad m_B \quad m_C \quad m_D \
  \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \
  A \quad B \quad C \quad D \quad A \quad B \quad C \quad D$

**MERGE**

- $m_A \quad m_C \
  \downarrow \
  m_A \quad m_B \quad m_C \
  \downarrow \quad \downarrow \quad \downarrow \
  A \quad B \quad C \

**Conclusion:** When augmenting must ensure can implement $ADD$, $SPlice$, $SPLIT$, $MERGE$!
Example 1: **Dynamic Order Statistics**

**Goal:** Would like to maintain Dynamic Set (insert, delete, search, successor ----) while additionally handling two new queries.

- **OS-SELECT\((i, S)\):** Returns \(i^{th}\) smallest element of \(S\).
- **OS-RANK\((x, S)\):** Returns rank of \((i.e., \# \text{el's} \leq x)\) \(x\) in \(S\).

**Note:** Can't do this with Dynamic Set operations efficiently.
Augmented 2-3 Tree

Additional Info: # leaves in each subtree

Structure of internal node:
$\text{OS-SELECT (i, node)}$

Let $\text{NUM}_{n} + \ldots + \text{NUM}_{j-1} < i \leq \text{NUM}_{n} + \ldots + \text{NUM}_{j}$.

Then return $\text{OS-SELECT (i-(NUM}_{n} + \ldots + \text{NUM}_{j}), child_{j})}$

Running time $= \Theta(\text{depth}) = \Theta(\log n)$. 

---
**OS-RANK** \((n, \text{node})\)

Let \(j\) be st.

\[
\max_{j-1} < x \leq \max_j
\]

Then return \(\left( \text{NUM}_1 + \ldots + \text{NUM}_{j-1} + \text{OS-RANK}(x, \text{child}_j) \right) \)

\[x\]

**Running time** = \(\Theta(\log n)\)

Conclude: Additional info does help answer queries.
Maintaining Additional Info?

No problem!

Assuming \( \text{Num}' \)'s known for children can compute \( \text{Num} \) for node.

\[
\text{Num}, + \text{Num}_2, + \text{Num}_3
\]
\[
\text{Num}', + \text{Num}'_2, + \text{Num}'_3
\]

Conclude: Can augment 2-3 tree to handle Dynamic Order Statistics.
Example

Num in brown
Max in green

3, 2, 3
4, 8, 14

1, 1, 1
1, 2, 4

1, 1
6, 8

1, 1
9, 11, 14

1 2 4 6 8 9 11 14

OS - SELECT (4) → 6
OS - RANK (6) → 9
INSERT (3)
New 2-3 Tree

Recompute NUM, Max for affected nodes
Example 2:

Interval Maintenance

- Data to be maintained = Set of "Intervals"
  
  Interval = \((low, high)\)

- Operations: INSERT, DELETE, Int-SEARCH
  
  \[\text{Int-SEARCH}(x, S) = \text{report (any) one interval } (l, h) \text{ in } S \text{ such that } l \leq x \leq h\]

Picture: $I-S(9)$ \(\Rightarrow\) returns (7,10) or (5,11)
Motivation

- Interval = Users of radio transmission medium + frequency range
- Users come in / drop off.
- Might want to know if a certain frequency is in use.

(More interesting would be... find all intervals overlapping \( x \), but that is harder to deal with ....)
Augmenting Methodology

- Pick Data Structure: 2-3 tree
- Key = \( l_i \)
- Additional Info: Clearly need to store \( h_i \) along with \( l_i \). What else?
- Thinking aloud: Need to have info that, depending on search point \( x \), eliminates left subtree or right tree of 2-ary node.
  - Right subtree eliminated if lowest key \( k \) in right tree is \( \geq x \).
  - \( \Rightarrow \) should store lowest key of subtree
- How could we eliminate left subtree?
  - If max high point in left subtree < x.
  - Should also store max high in subtree.

- Does this suffice? YES!
  
  Will see!
Say we're searching for $x$
and we reach node

\[ \text{MAX HIGH}_1, \text{MAX HIGH}_2, \text{MIN LOW}_1, \text{MIN LOW}_2 \]

Case 1: $x < \text{MIN LOW}_2$

$\Rightarrow$ Recurse on child$_1$

(Since every interval in child$_2$ has $\text{HIGH} > \text{LOW} > x$, $x$ can't overlap with $x$)
**Case 2:** \( \max \, \text{HIGH}_1 < x \)

\[ \Rightarrow \text{Recurse on child}_2 \]

(No interval in child\(_1\) overlaps with \(x\))

... Not Exhaustive ...

**Case 3:** \( \min \, \text{LOW}_2 \leq x \leq \max \, \text{HIGH}_1 \)

**Claim:** Should recurse on child\(_1\)

**Why?** ... Not the same reasoning as before

This time: Because there exists an interval in child\(_1\), in particular the one with \( \text{HIGH} = \max \, \text{HIGH}_1 \) that overlaps with \(x\).
Formally: let \((\text{Low, High})\) be interval in child with
\[\text{High} = \max \text{ High}_1\]

Then
\[\text{Low} \leq \min \text{ Low}_2 \leq \alpha\]
\[\uparrow \quad \uparrow\]
\[\text{Search} \quad \text{Case assumption}\]
\[\text{Property}\]

And
\[\alpha \leq \max \text{ High}_1 = \text{High}\]
\[\uparrow\]
\[\text{Case assumption}\]

So \(\text{Low} \leq \alpha \leq \text{High}\)
Finally: Car maintain MAX HIGH; MIN LOW
by local calculations...

Conclusion: Augmented 2-3 trees
& Augmented Data Structures
Can solve many new problems!