RED-BLACK TREES

(Notes by Henri Mertens)
Binary search trees

Nodes are colored: \( \text{color}[x] \in \{\text{red, black}\} \)

Nodes have a rank: \( \text{rank}[x] \geq 0 \)  (related to height)
Rules:

- **External nodes of rank 0**: Black.
- For black: \( \text{rank}[\text{parent}[x]] = \text{rank}[x] + 1 \)
- For red: \( \text{rank}[\text{parent}[x]] = \text{rank}[x] \)
  
  \( \text{color}[\text{parent}[x]] = \text{black} \)

\( m \) internal modes (with keys)

\( \text{Not allowed} \)
PROPERTIES

$m$ internal nodes
$m+1$ external nodes

If a node has rank $r$, then

1. $2 \leq m + 1 \leq 4^r$
2. $r \leq R \leq 2r$

Therefore, for the root,

$L \leq 2r \leq 2 \log_2 (m + 1)$.

$\rightarrow$ Red-Black trees are balanced search trees.
Proof of $\text{(i)}$:

- $\tau = 0$:
  
  \[ 2^\tau = 1 \leq n+1 = 1 \leq 4^\tau. \]

- For general $\tau > 0$, assuming $\text{(i)}$ for smaller values:

There exist modes of rank $\tau - 1$ in left and right subtrees, so the number of external nodes, $n+1$, is

\[ \geq 2^{\tau-1} + 2^{\tau-1} = 2^\tau. \]

Also, we can find at most 4 modes of rank $\tau - 1$ that are not ancestors of each other. So,

\[ \text{# External nodes} = n+1 \leq 4^{\tau-1} + 4^{\tau-1} + 4^{\tau-1} + 4^{\tau-1} = 4^\tau. \]

Proof of $\text{(ii)}$: A similar induction (exercise!).
The 2-3-4 tree view of a red-black tree

**Pods:**

Place all black nodes of rank r at same level.
OPERATIONS

INSERT, DELETE as for binary search trees
Fix to restore the red-black tree property

Total time $O(d \log n)$
Fix (after INSERT)

Before:

After:

Let \( x \) be (a pointer) to the new mode.
Increase rank of grandparents recursively until a black parent is reached

OR

A red parent / Black uncle
while \( p[x] \neq \text{nil}, p^2[x] \neq \text{nil} \),
\[ \text{color}[p[x]] = \text{red}, \text{color}[v[x]] = \text{red} \]
do:

\[ x \leftarrow p^2[x] \]
\[ \text{color}[\text{left}[x]] = \text{black} \]
\[ \text{color}[\text{right}[x]] = \text{black} \]
\[ \text{rank}[x] \leftarrow \text{rank}[x] + 1 \]
\[ \text{color}[x] \leftarrow \text{red} \]

\[ \text{case} \]
\[ p[x] = \text{nil} : \{ \text{do nothing} \} \]
\[ p[x] \neq \text{nil}, p^2[x] = \text{nil} : \]
\[ p^2[x] \neq \text{nil}, \text{color}[p[x]] = \text{black} : \{ \text{do nothing} \} \]
\[ p^2[x] = \text{nil}, \text{color}[p[x]] = \text{red}, \text{color}[v[x]] = \text{black} : \]

\[ \text{perform local fix} \]

if root is \( \circ \), then increase rank root by 1 to get

// a is the new node //
\[ \text{write} \]
\[ p[a] \]
\[ p^2[a] \]
\[ v[a] \]

uncle

\[ \text{while} \]
\[ \text{if} \]
**Case 1**

- **Fix**: 

  - **Root Fix**
    - **Case 2**
      - **DONE**

  - **Root**: 
    - **x**
    - **r**

  - **Rank** \([p(x)]\) + = 1

  - **Only way to increase the rank of the root**

- **Bank** \([\text{plate} \text{t} = I] \text{te}\) \(\text{Elon}\) \(\text{only way to root})
LOCAL Fix

\[ x \]

or

\[ a \quad \text{or symmetric} \]
Deletion

Similar principle: Standard deletion + Fix tree from bottom up by demotions

Lazy deletion: Leave node as is. Mark it "deleted".

When # of marked nodes is > 50%, store the nodes in an ordered list L. Make a fresh red-black tree in time $O(n)$. (Exercise.)

Also see question 2 of assignment 4.
ADDITIONAL OPERATIONS: SPLIT, JOIN

\[ T \]

SPLIT \((t, R)\):
- Split the tree into \( T_1 \) and \( T_2 \).
- Ensure keys of \( T_1 \) are less than keys of \( T_2 \).

JOIN \((t_1, t_2)\):
- Combine \( T_1 \) and \( T_2 \) to form a new tree.

Time: \( O(\log n) \)
**JOIN** \((t_1, t_2)\)

**Step 1:** \(\alpha = \text{MAXIMUM}(t_1)\)

**Step 2:** \(\text{DELETE}(A_1, \alpha)\)

**Step 3:** Find highest node of rank \(r_2 = \text{rank}(t_2)\) on the right roof of \(t_1\), say \(z\).

**Step 4:** Create \(t_1\).

**Step 5:** Update the tree starting at \(x\).
Split $(A, R)$

Step 1.

Decompose $T$ into the components of $T_1, T_2$

$(Time = O \log n)$

Make $T_1$

Make $T_2$

Decompose $T$

value $k$

Tinto

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$

$q\#$

$D$

$D$

value $k$

root

Make $T_1$

External mode

value $R$

Decompose $T$
Exercise. Convince yourself that the total time is $O(bpm)$. 
Lists: \textsc{concatenate} / \textsc{sublist}

- \textit{List}
  - \textit{Sublist} → \textsc{split}
  - \textit{List1} \quad \textit{List2} → \textsc{join}

All operations are \(O(\log n)\).

Implement by red-black tree. Must keep track of position in the list (see next chapter).

Recall: Arrays have \(O(n)\) \textsc{join}

Linked lists have \(O(n)\) \textsc{split}