Binary counter (ℓ-bit)

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
⋮

digits changed = (# trailing 1's) + 1
w.c. ℓ

Aggregation

n × Increment

bit changes $\leq n + \frac{n}{2} + \frac{n}{4} + \dots$

$$\leq \sum_{i=0}^{\ell-1} \frac{n}{2^i} = n \cdot \sum_{i=0}^{\ell-1} \frac{1}{2^i} \leq 2n$$

$$\leq \sum_{i \geq 0} \frac{1}{2^i} = 2$$

↳ total time is $\mathcal{O}(n)$

in binary: $1.111\ldots = 10.00\ldots$

Coin method

"time coins"

1 \$ ≈ time to change a single bit

for every Inc, I ask for 2 \$

Invariant: We have 1 \$ on every 1 bit.

Single Inc: given 2 \$

exactly one $1 \to 0$ releases 1 \$, which pays for the change

$0 \to 1$ spend 1 \$ on the change, leave 1 \$ on the 1

Potential Method

Sequence of operations: $O_1 \dots O_m$

Real costs: $R_1 \dots R_m$

Amortized costs: $A_1 \dots A_m$

↳ we need: $\sum_{i=1}^{m} R_i \leq \sum_{i=1}^{m} A_i$

Potential: Φ_i after executing $O_1 \dots O_i$

$\Phi_0 \dots \Phi_m$

$$\Phi_i := \sum_{j=1}^{i} (A_j - R_j) = \left(\sum_{j=1}^{i} A_i\right) - \left(\sum_{j=1}^{i} R_j\right)$$

We want: $\Phi_0 = 0$

$\forall i \ \Phi_i \geq 0$

potential difference

$$\Delta\Phi_i := \Phi_i - \Phi_{i-1} = A_i - R_i$$

> 0 if saving time

< 0 if spending saved time

Examples:

① **binary counter**

R_i := # bit changes

$A_i := 2$

Φ_i = # 1 bits

$\Delta\Phi_i$ = # 1 gained = # 0→1 − # 1→0

= 2 − # bit changes

② **shrinking array**

R_i = # items moved (cost of reallocation)

$A_i := 2$

Φ_i := # operations since the last reallocation

Also: $\Phi_i = A_i - R_i$

We can do: analyse R_i, define Φ_i → obtain $A_i := R_i + \Delta\Phi_i$

We have: $\sum_{i=1}^{m} A_i = \sum_{i=1}^{m} R_i + \sum_{i=1}^{m} \Delta\Phi_i$

$\Delta\Phi_i = \Phi_i - \Phi_{i-1}$

$(\Phi_1 - \Phi_0) + (\Phi_2 - \Phi_1) + (\Phi_3 - \Phi_2) + \dots + (\Phi_m - \Phi_{m-1})$

telescoping sum $= \Phi_m - \Phi_0$

$(\Phi_i + c) - (\Phi_{i-1} + c) = \Phi_i - \Phi_{i-1}$

$$\sum_{i=1}^{m} p_i = \left(\sum_{i=1}^{m} A_i\right) + \Phi_0 - \Phi_m$$

this happens whenever $\Phi_0 = 0$ & $\forall i\ \Phi_i \geq 0$

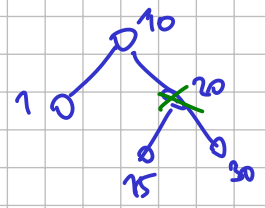
ok if ≤ 0

↳ we get $\sum_i p_i \leq \sum_i A_i$

To recap:

Amortized analysis:
1. Establish real costs (& choose units)
2. Define potential — or upper bound for
3. Calculate amortized costs
4. Verify that $\Phi_0 \leq \Phi_m$

Handling deletions



instead: replace items by tombstones

↓

leads to degradation (too many tombstones)

↓

rebuild the structure & remove all tombstones

$\Theta(n)$

↑

including tombstones

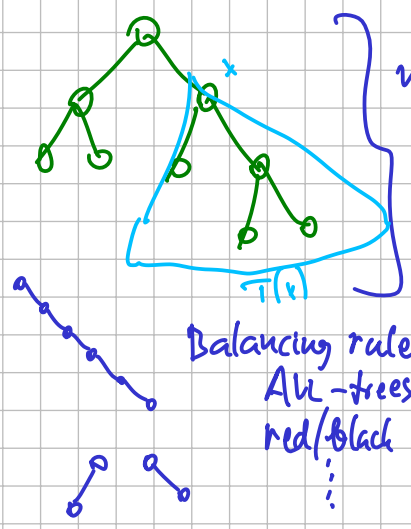
Idea: After a rebuild of size n, next rebuild in n operations.

↳ cost of rebuild is $O(1)$ amort.

} if 1 operation saves $\Theta(1)$ time, this is enough to pay for the rebuild

---

Lazily balanced binary trees - Weight-balanced trees



we want height $O(\log n)$

Balancing rules:
AVL-trees
red/black trees
⋮

Df: For a node x in a binary tree:

$T(x)$:= subtree rooted at x $\quad T(\emptyset) := \emptyset$

$\ell(x), r(x)$:= left/right child or $\emptyset$

$s(x) := |T(x)|$ ... # descendants of x $\quad s(\emptyset) = 0$

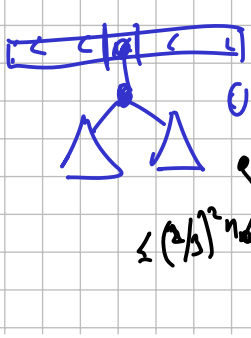
$n :=$ # nodes size of subtree

Perfectly balanced tree

For every node x: $\left|\, s(\ell(x)) - s(r(x)) \,\right| \leq 1$.

↳ a tree can be rebalanced to perfect balance in $\Theta(n)$ time.

↳ relax: ratio between 1:2 and 2:1



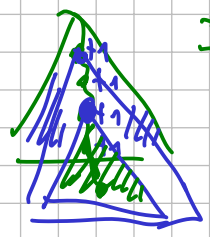
Perfectly balanced
$O(n)$ time

Df: A node x is in balance $\equiv$ for every child c of x, $s(c) \leq \frac{2}{3} \cdot s(x)$

Tree is balanced if ∀ node ↗

↳ This guarantees $O(\log n)$ height of the tree.

Keeping tree in balance (Inserts only)



In every node, I remember $s(x)$

Insert: increase $s(x)$ by 1 on the path to the root & check balance

└ if out of balance: rebuild the highest out-of-balance subtree } takes time $\Theta(s(x))$

everything else is $O(\log n)$

Potential:

For a node x: contribution $\varphi(x) := |s(\ell(x)) - s(r(x))|$

$\Phi := \sum_x \varphi(x)$ Fix: make this 0 if it were 1

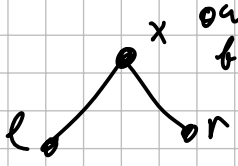
easy: $\Phi_i \geq 0$

initially empty tree, so $\Phi_0 = 0$.

in a perfectly balanced tree, all contribs are 0

① Adding new leaf: real cost 1
but $\Delta\Phi$ is up to $\log n$ } → amort. cost $O(\log n)$

② Rebalancing a subtree:



WLOG $s(\ell) > 2/3 \cdot s(x)$
So $s(r) < 1/3 \cdot s(x)$ } so $\varphi(x) \geq 1/3\, s(x)$

this is released by rebalancing

it can pay the cost $\Theta(s(x))$ of rebalancing

all $\varphi(y)$'s here become zero

In total, $-\Delta\Phi \geq 1/3\, s(x)$.

→ amort. cost of rebalancing is zero!

Conclusion: Insert has amort. complexity $O(\log n)$.

Find has worst-case —"—