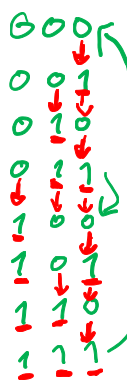
Coin method

Ex: binary counter: INC (increment), ℓ bits, unit cost = 1 bit flip



Obs) worst case

Obs: bit i changes once in every 2^{i-1} operations

n INCs $\Rightarrow$ by aggregation $\sum_{i=1}^{\ell} \left\lfloor \frac{n}{2^{i-1}} \right\rfloor \le n \sum_{i=1}^{\infty} \frac{1}{2^{i-1}} = 2n$

$\Rightarrow$ $\Theta(1)$ amortized time

paying in advance: 2 coins for each operation (bitcoins)

(1 coin for $0 \to 1$, 1 coin stored on the next 1 for $1 \to 0$, use the stored coins)

$\Rightarrow$ $\Theta(1)$ amortized time

Potential method (general approach)

goal: compensate "expensive" operations by "cheap" ones

n operations, real cost R_i (upper bounds), amortized cost A_i (our claim)

after i operations: $\Phi_i := \sum_{j=1}^{i} A_j - \sum_{j=1}^{i} R_j$ potential "bank account"

$\Phi_i \ge 0$ "never owe time"

$\Phi_0 = 0$ (by definition)

potential difference $\Delta\Phi_i := \Phi_i - \Phi_{i-1} = A_i - R_i = \begin{cases} \ge 0 & \text{saves time} \\ = 0 & \text{pays exactly} \\ < 0 & \text{spends time} \end{cases}$ ith operation:

Ex: binary counters

$\Phi_i = \#1\text{'s in } (i)_2$

$A_i = 2$, $R_i = 1 + \underbrace{\#\text{trailing 1's}}_{t}$

$\Delta\Phi_i = 1 - t$

Ex: flexible array

$\Phi_i = \#$ operations since last reallocation

$A_i = 2$ (1 for APPEND/REMOVE, 1 for potential)

$R_i = \begin{cases} 1 & \text{no realloc} \\ 1 + \Theta(c) & \text{with realloc} \end{cases}$ (rescale the potential)

What is amortized cost?

$A_i := R_i + \Delta\Phi_i$ (when potential defined first)

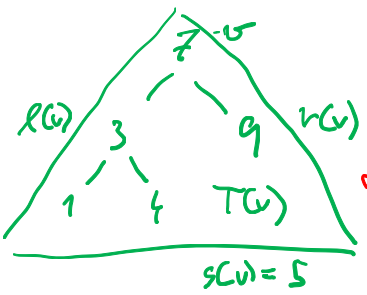
after n operations: $\sum_{i=1}^{n} A_i = \sum_{i=1}^{n} (R_i + \Phi_i - \Phi_{i-1}) = \sum_{i=1}^{n} R_i + \underbrace{\Phi_n - \Phi_0}_{\ge 0}$ telescopic sum

$\Rightarrow$ Σ real costs bounded by Σ amortized costs ✓ ← we need

Summary: 1) choose unit cost (time/space linear in it)
"recap" 2) estimate real costs (upper bounds)
3) choose an appropriate potential Φ (i.e. counting something in the structure)
4) show that $\Phi_m - \Phi_0 \geq 0$ (never go bankrupt)
5) aim for $A_i := R_i + \Delta\Phi_i$ to be small

Lazily balanced BSTs (example for amortized complexity)

idea: let the structure "degrade", but occasionally rebuild it (locally)



not perf. but balanced

Notation: for a node v in BST
$T(v)$ ... subtree rooted at v (including v)
$s(v)$ ... size of v := # nodes in $T(v)$
$\ell(v), r(v)$ ... left, right child of v ($\emptyset$ if child missing)

Def: BST is perfectly balanced if $|s(\ell(v)) - s(r(v))| \leq 1$ for every v. (ratio 1:1 (almost))

☼: sorted n keys ⇝ perfectly balanced BST can be built in $O(n)$ time.

Def: a node v is in balance if $s(c) \leq 2/3\, s(v)$ for each child c of v. (between 1:2 and 2:1)
BST is balanced if all its nodes are in balance.

☼: The height of a balanced tree with n nodes is $O(\log n)$.

Pf: $h \left\{ \begin{matrix} n \\ \vdots \\ 1 \end{matrix} \right. \leq 2/3\, n$ $\quad 1 \leq \left(\frac{2}{3}\right)^h n \Rightarrow h \leq \log_{2/3}(1/n) = \log_{3/2} n = O(\log n)$ ∎

INSERT (lazy): standard INSERT in BST ← keep $s(v)$ at each v
update $s(v)$ in each v on the path to the root (on the path)
if all nodes in balance, stop.
else REBUILD($T(v)$) at the highest out-of-balance node v

perfectly balanced BST in time $\Theta(s(v))$

Thm: INSERT works in $O(\log n)$ amortized time.

Pf: potential $\Phi := \sum_v \varphi(v)$ (sum of contributions)

$$\varphi(v) := \begin{cases} |s(\ell(v)) - s(r(v))| & \text{if at least 2} \\ 0 & \text{otherwise} \end{cases}$$ (think why)

amortized cost of INSERT:

a) without rebuild

$O(\log n) + O(\log n) = O(\log n)$

↑ real cost ↑ change of potential: each $\varphi(v)$ increases by ≤ 2. on the path

b) rebuild at node v

w.l.o.g. $s(\ell(v)) > 2/3\, s(v) \Rightarrow s(r(v)) < 1/3\, s(v)$

$\Rightarrow \varphi(v) > 1/3\, s(v)$ this drops to 0 in Φ

$\Rightarrow \Delta\Phi \le -1/3\, s(v)$ potential drops by at least $1/3\, s(v)$

amortized cost of REBUILD $= \Theta(s(v)) + c \cdot \Delta\Phi \le 0$

(i.e. for free)

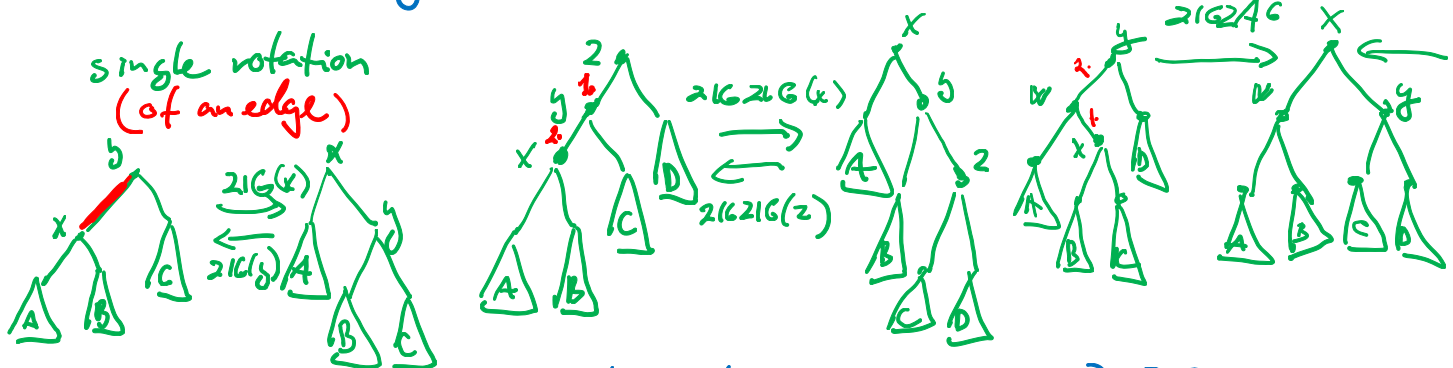
↑ real cost for a suitable constant c. (rescaling the potential, amortized costs)

$\Rightarrow O(\log n)$ amortized time for INSERT ∎

Splay trees (intro)

self-balancing BST with $O(\log n)$ amortized costs

SPLAY(x) ... bring x to the root, preferring double rotations:



ZIG only if x child of the root, otherwise ZIGZIG or ZIGZAG

IDEA: always splay the lowest visited node (at the end of each operation).