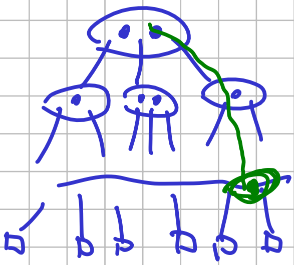
(a,b)-tree



height ~ log n

w.c. complexity

Find $O(\log n)$

Ins,Del $O(b \cdot \log_a n)$

external nodes

$a \geq 2$ $b \geq 2a-1$

between a and b children

a-1 b-1 keys

Amortized Analysis

Warmup seq. of m Inserts to an initially empty tree

Thm: Total # of modified nodes is $O(m)$.

Proof: $\# \text{modifications} = O(m) + O(\# \text{splits})$

initial step the rest

$\# \text{splits} \leq \# \text{nodes at the end} \leq \# \text{keys at the end} = m$

General case: Seq. of m Inserts and Deletes starting with an empty tree.

Thm: # modifications is $O(m)$ for $b \geq 2a$.

we will prove for $b = 2a$.

Proof: Use potential technique.

$$\Phi := \sum_{v \text{ node}} f(\underbrace{\# \text{keys in } v}_{})$$

$f: \mathbb{N} \to \mathbb{N}$

a-2 to 2a~~-1~~

underfull overfull

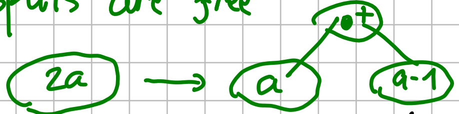
initially $\Phi = 0$
always $\Phi \geq 0$

idea: Split/Merge has 0 amort. cost (real cost := 1)

Requirements on f:

① $\exists c : \forall k \; |f(k) - f(k-1)| \leq c$

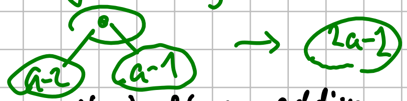
② splits are free



releasing $f(2a)$ adding at most $f(a) + f(a-1) + c$

$f(2a) \geq f(a) + f(a-1) + c + 1$

③ merges are free



releasing $f(a-2) + f(a-1)$ adding $f(2a-2) + c$

$f(a-2) + f(a-1) \geq f(2a-2) + c + 1$

Use this f:

k	a-2	a-1	a	....	2a-2	2a-1	2a
f(k)	2	1	0	....	0	2	4

Verify: ① ok with c=2 ② $4 \geq 0+1+2+1$

③ $2+1 \geq 0+2+1$

Am. cost of Insert = $O(c)$ + 0 per Split

initial insertion

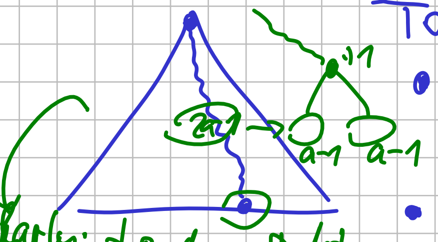
Am. cost of Delete = $O(c)$ + 0 per Delete + $O(c)$

init. del. stealing key

always a constant

Q.E.D.

Parallel program
↓
need locking



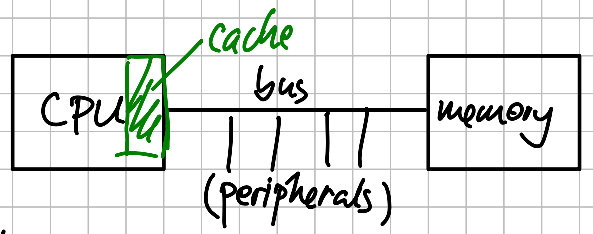
similarly: top-down Delete

Top-down Insert for $b \geq 2a$:

- when finding the place for the new node, split nodes with 2a-1 keys
- when adding the new item, we are sure we have space for it

pre-emptive splitting

Caching



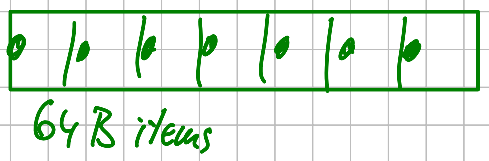
typical parameters:

L1 cache	32 KB	in 64B blocks
L2 cache	256 KB	
L3 cache	4 MB	

1 cycle of 1GHz clock = 1ns = 10^{-9} s

speed of light ≐ $3 \cdot 10^{8}$ m/s

↳ in 1 cycle, light travels 30 cm



I/O model (ext. mem. model)

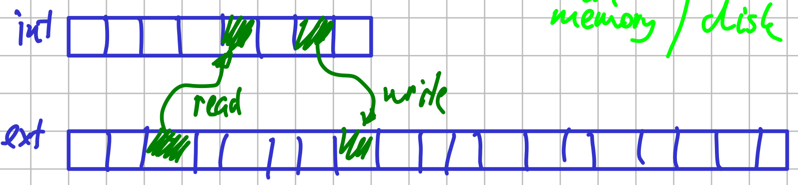
memory — internal (cache / main mem.) — of limited size, we can compute here (RAM-like)

memory — external (main memory / disk) — potentially infinite, split to blocks

instructions for transfer of blocks between int. and ext. mem.

parameters: B = block size

M = size of int. memory



Complexity measures: time

space

I/O (communication)

↳ typically count just reads

Caching models

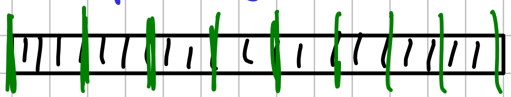
- int. and ext. memory, params B and M
- program addresses data in ext. mem.
- there is a cache controller which brings the data to int. mem. as needed

↳ we assume that the controller is optimal (it minimizes the I/O complexity)

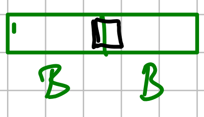
Cache-aware - program knows B, M

Cache-oblivious - it doesn't

Warm-up: Array Scan size N



stored in ⌈N/B⌉ consecutive blocks



I/O model: #reads = ⌈N/B⌉

we need $M \geq B$

cache-aware model: #reads = ⌈N/B⌉

cache-oblivious model: we cannot ensure alignment

#reads = ⌈N/B⌉+1 in w.c.

Conclusion: #reads ∈ $O(\underbrace{N/B}_{\text{arbitrarily small}} + 1)$

for infinitely many (N,B) $O(N+1) = O(N)$