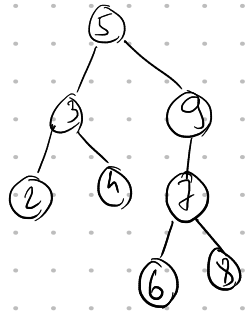
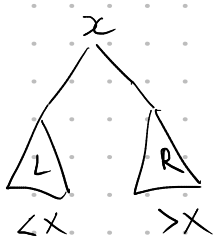
Online problems

- caching/paging
- list update
- binary search trees (TODAY)

e.g.





(assume all keys distinct)

Store an ordered set of size $\Theta(n)$

operations: - search
- insert
- delete
- ...

$O(\log n)$, as long as tree is "approximately balanced"

↓

depth $O(\log n)$

e.g. AVL
red-black-trees
(2,4)-trees

need nontrivial bookkeeping/maintenance

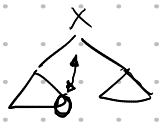
$O(\log n)$ is best possible.

tree of n nodes, some leaf has depth $\geq \log_2 n$

(How to beat $\log n$ - cost using binary search trees?)

recall: - search(x): follow search path from root to x (or succ/pred if x missing)

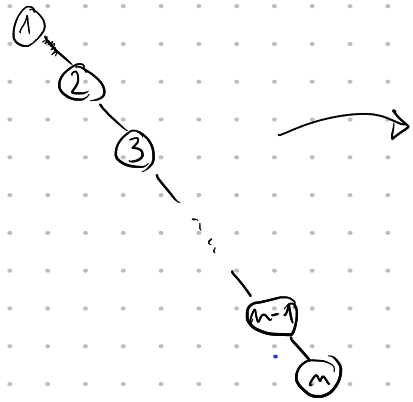
- insert(x): follow search path to succ/pred., attach x as child

- delete(x): search for x
  - if x has 0 or 1 child, just cut x out
  - if  swap x with predecessor, then cut x out.

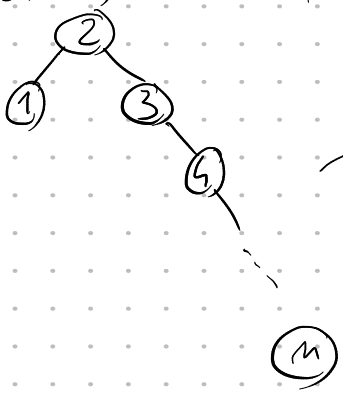
When / how can we beat $O(\log n)$?

e.g. (1) $R = 1, 2, 3, \dots, n$. → sequence of searches

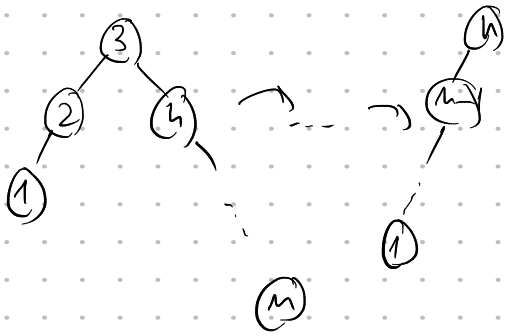
search(1) → cost: 1+1



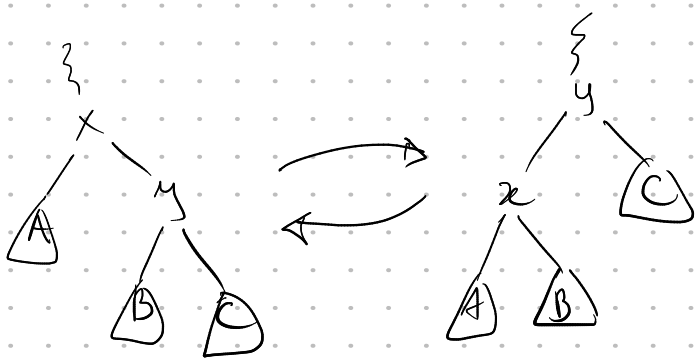
search(2) → cost: 1+1



cost: 2



Rotation:

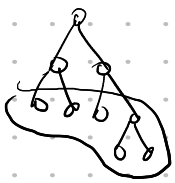


cost: 1

$R = 1, 2, \dots, n$

total search cost for serving R is $O(n)$.

(Note: any fixed tree can achieve at best $O(n \log n)$ for this sequence, or for any other permutation-sequence)



$\geq \log_2 n - 1$

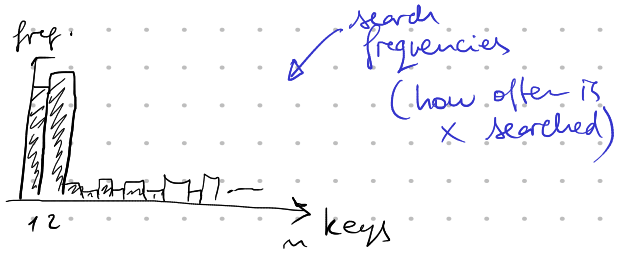
→ most nodes have depth $\sim \log_2 n - c$

$OPT(R) = O(n)$

(In this case we could do better bec. we exploited special ordering of search-sequence)

e.g. (2) length m, n diff. keys $m \gg n$

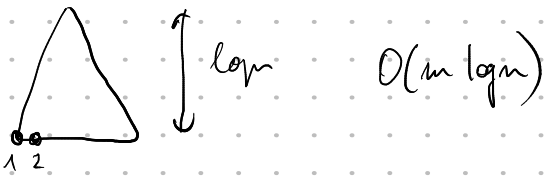
$R = 1, 2, 1, 5, 1, 2, 2, 3, 1, 1, \ldots$



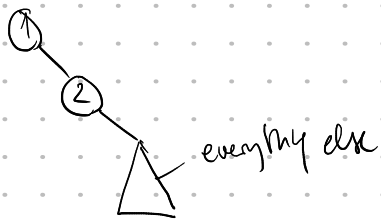
freq.-distrib. "very uneven"

(in this example the ordering of R did not play a role)

Cost of serving R in a balanced tree



Build optimal fixed (static) BST for seq. R



cost of serving R may be $o(m \log n)$

could even be $O(m)$

Finding best static tree – Dynamic Programming $O(n^2)$ → This assumes that R is known in advance.

(OPTIMUM BST) [Knuth, 1971]

Question: Can we achieve similar running times without knowing R in advance?

→ Would like an online competitive BST.

→ efficient on R (whenever possible)

Self-adjusting binary search tree (Splay tree)

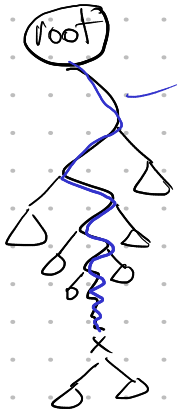
[Sleator-Tarjan, 1983]

Idea:

- maintain a simple BST (no bookkeeping, no structural information, not always balanced) → simple
- after each operation, do some restructuring of tree,
  - such as to prepare for future accesses.
  - even after search. (This is different from classical balanced BST like AVL, that restructure only after insert/delete. More similar to list update / paging)
- guarantees are amortized
- online, "adaptive" strategy, a form of learning

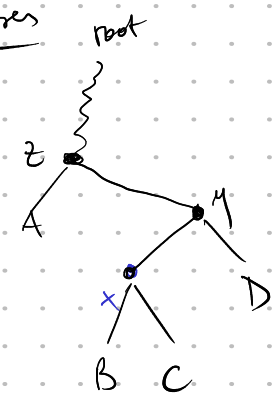
Splay-tree

- simple BST
- search(k):
  - rotate x up to the root
  - restructure search path

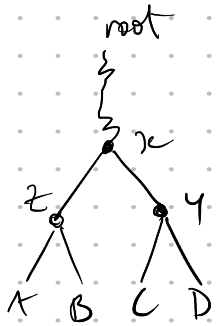


Splay operation

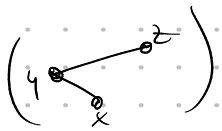
3 cases



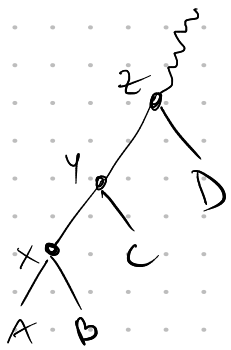
(zig-zag)



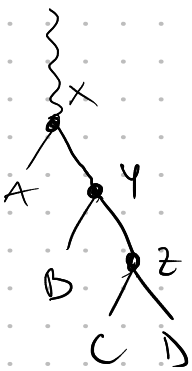
(symetric case:



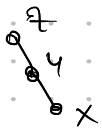
)



(zig-zig)

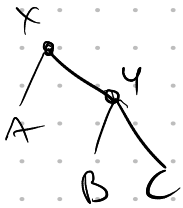


(symetric case:



)

(zig)



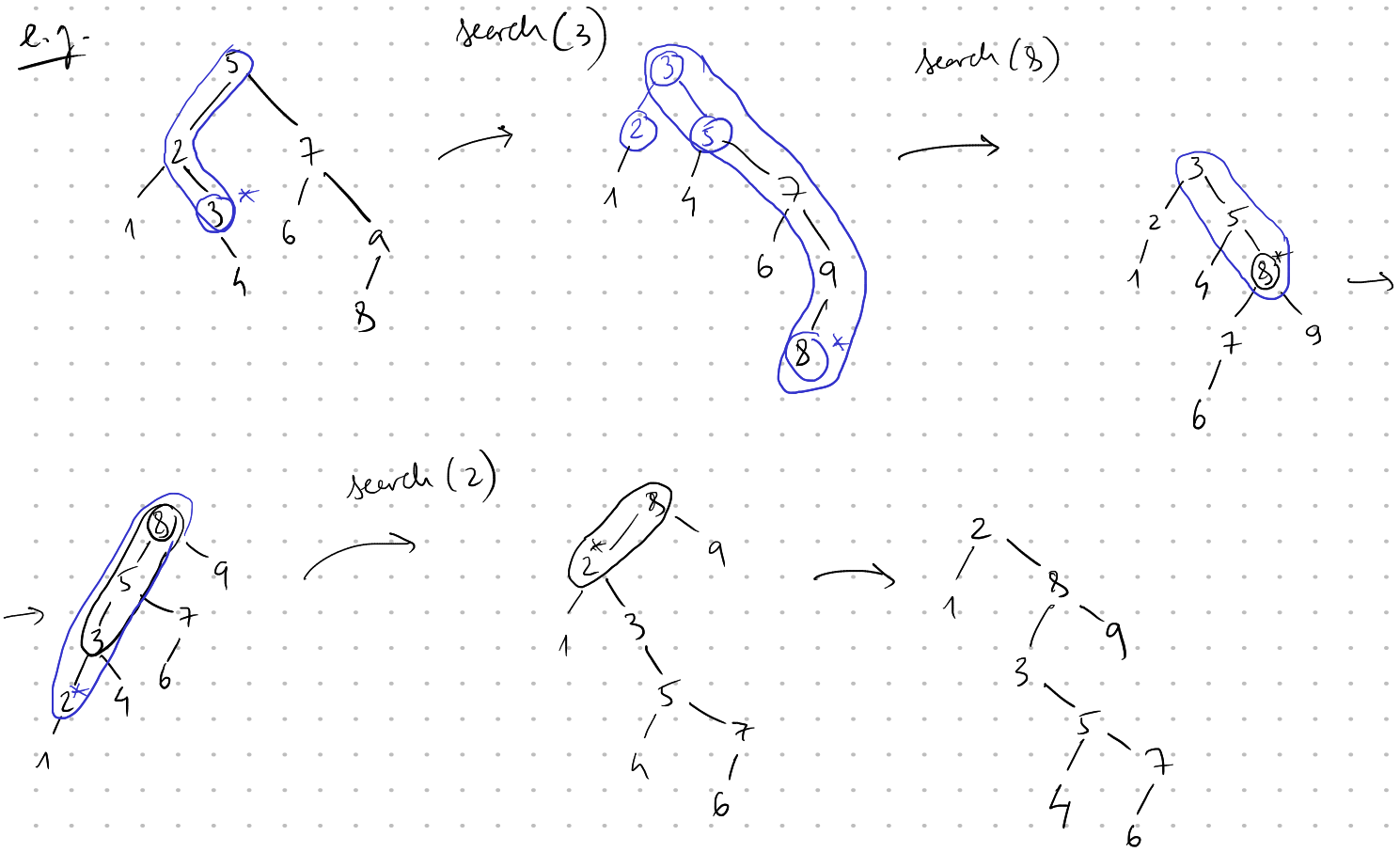
(symetric case:

)

- search(k):
  normal BST-search
  splay(x)

obs. Cost of splaying $\sim$ cost of searching
$=$
length of search path

e.g.



(tree not always balanced)

→ $O(\log n)$ amortized time

→ Adaptive behaviour on various search sequences.

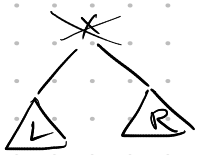
Other operations

insert(x)

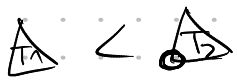
- insert normally as in BST
- splay(x)

delete(x)

- splay(x)
- remove root
- join(L, R)



join(T_1, T_2)

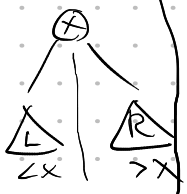


- splay(max(T_2))
- link T_1 as left child of root

alternative method

split tree at x

link L, R as children of x



split at x

splay(x), remove root

Known properties of Splay trees

$R = r_1, \dots, r_m$ sequence of searches (typically $m \gg n$)

$r_i \in \{1, \dots, n\}$ for all $i = 1, \dots, m$

Thm 1. Serving R in a splay tree (with arbitrary initial tree)
has total cost $O((m+n)\log n)$
($O(m \log n)$ if $m > n$)
[logarithmic amortized cost]

Thm 2. Let T be an arbitrary BST over $\{1, \dots, n\}$
Serving R in a splay tree (with arbitrary i.t.)
has total cost $O\left(\sum_{i=1}^{m} d_T(r_i) + \underbrace{m + n\log n}_{\text{ignore if } m \gg n}\right)$

$r_1, \dots, r_m$

T
reference tree

[Static optimality theorem]
(Splay tree is competitive with best fixed tree for seq. R)

Thm 2 $\Rightarrow$ Thm 1
(Choose T to be a balanced tree)

Thm 3. Let $R = 1, 2, \dots, n$
Cost of R using splay is $O(n)$ with arbitrary initial tree.
[Sequential access thm.]

Thm 4.
$R = r_1, \dots, r_m$
Cost of serving R using splay (arbitrary i.t.) is

$$O\left(\sum_{i=1}^{m} \log t_i + m + m\log n\right)$$

"temporal locality"

t_i: # different elements searched since previous time we searched r_i, or $t_i = n$ if r_i searched first time.

$R = 1, 5, 3, 5, 4, 2, 5, 3, \ldots$

$\log n \quad \log n \quad \log n \quad \log 2 \quad \log n \quad \log n \quad \log 3 \quad \log 4$

[Working set thm]

Thm 4 $\Rightarrow$ Thm 1

Thm 5 $R = r_1, \ldots, r_m$ "spatial locality"

cost of splay:

$$O\left(\sum_{i=1}^{m} \log(|r_i - f| + 1) + m + m \log n\right) \quad (\checkmark)$$

for arbitrary fixed f.

[static finger thm]

Thm 6

cost of splay

$$O\left(\sum_{i=1}^{m-1} \log(|r_{i+1} - r_i| + 1) + m\right)$$

[Dynamic finger thm]

[Cole 90s]

80+ pages

Thm 6 $\Rightarrow$ Thm 3

Conjectured properties

Deque property

R sequence of search, insert, delete operations, only on current min/max in tree

cost of splay $O(n)$.

Known $O(n \cdot \alpha(n))$

Dynamic optimality

Major open question of theoretical computer science

[Sleator-Tarjan, 1983]

Cost of splay on R is $O(OPT(R))$

$\Updownarrow$

smallest cost for serving R using any BST with rotations, knowing R in advance.

In other words: splay is $O(1)$-competitive

Thm 1

Splay tree has $O(\log n)$ amortized cost per operation.

Proof

$R = r_1, \dots, r_m$, n size of tree

c_i : cost of search(r_i)

↳ depth of r_i before search

a_i : amortized cost of search(r_i)

$$a_i = c_i + \phi_i - \phi_{i-1}$$

$\phi_i - \phi_{i-1}$: pot. incr. due to i-th search

ϕ_i = pot. after i-th search

Potential fct.

$s(x)$: size of subtree rooted at x

= # nodes (incl. x)

"rank"

$$r(x) = \log_2 s(x) \geq 0$$

$$\phi(T) = \sum_{x \in T} r(x)$$

"sum-of-logs pot."

Obs.

$\phi(T) \geq 0$

$\phi(T) \leq n \log n$

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} c_i + \phi_m - \phi_0$$

total amort. cost

total act. cost

$$\sum_{i=1}^{m} c_i \leq \sum_{i=1}^{m} a_i + \phi_0 - \cancel{\phi_m} \leq \sum_{i=1}^{m} a_i + n \log n$$

Claim $a_i \in O(\log n) \quad \forall\, i = 1, \dots, m$

(observe that Thm 1 follows from Claim)

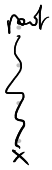
Proof

$$a_i = c_i + \Delta\phi$$

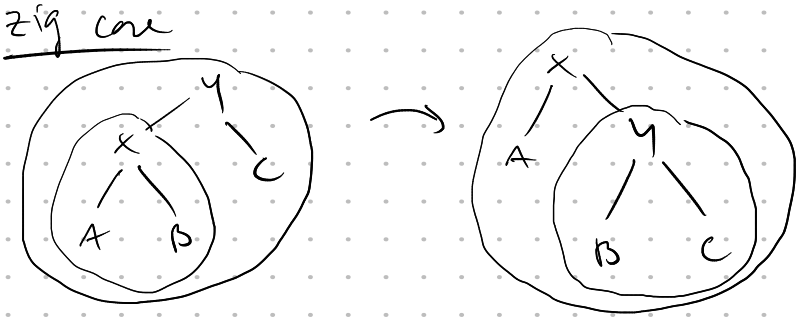
c_i : length of search path

$\Delta\phi$: incr. in pot.

Split $\Delta\phi$ into separate parts due to individual zig, zig-zig, zig-zag } operations



Zig case



only $r(x)$, $r(y)$ change

r^+ : after

r^- : before

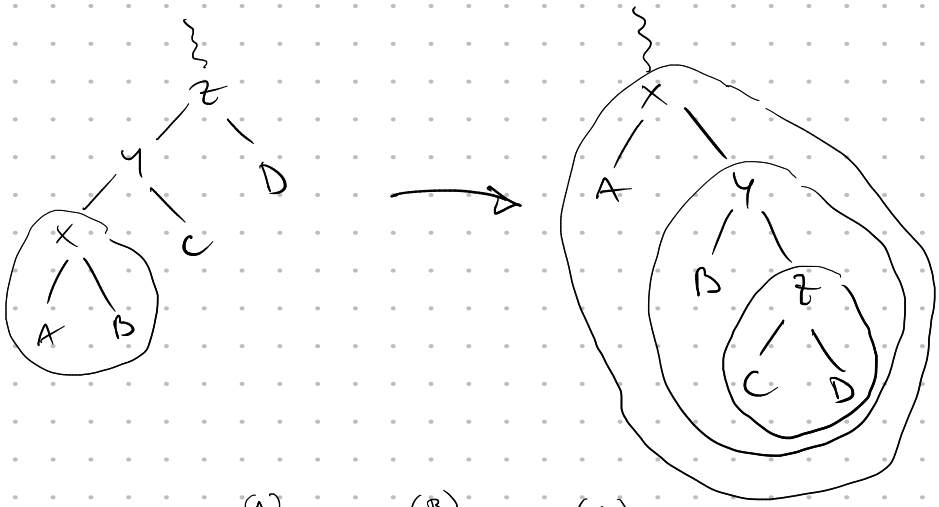
$$\Delta\phi = r^+(x) + r^+(y) - r^-(x) - r^-(y)$$

Obs. $r^+(y) \le r^+(x)$

$r^-(x) \le r^-(y)$

$$\boxed{\Delta\phi \le 2(r^+(x) - r^-(x))}$$

Zig-zig case



$$\Delta\phi = r^+(x) + r^+(y) + r^+(z) - r^-(x) - r^-(y) - r^-(z)$$

$$\le 3(r^+(x) - r^-(x))$$

(A) (B) (C)

Obs. $s^+(x) \ge s^+(z) + s^-(x)$

$$2r^+(x) \ge 2 + r^+(z) + r^-(x)$$

$$r^+(z) \le 2r^+(x) - 2 - r^-(x) \quad (*)$$

$$\Delta\phi = r^+(x) + r^+(y) + r^+(z) - r^-(x) - r^-(y) - r^-(z)$$

$$\le 3r^+(x) - 2r^-(x) + r^+(y) - r^-(y) - r^-(z) - 2$$

$$\boxed{\Delta\phi \le 4(r^+(x) - r^-(x)) - 2}$$

$A \ge B + C \;/\;()^2$

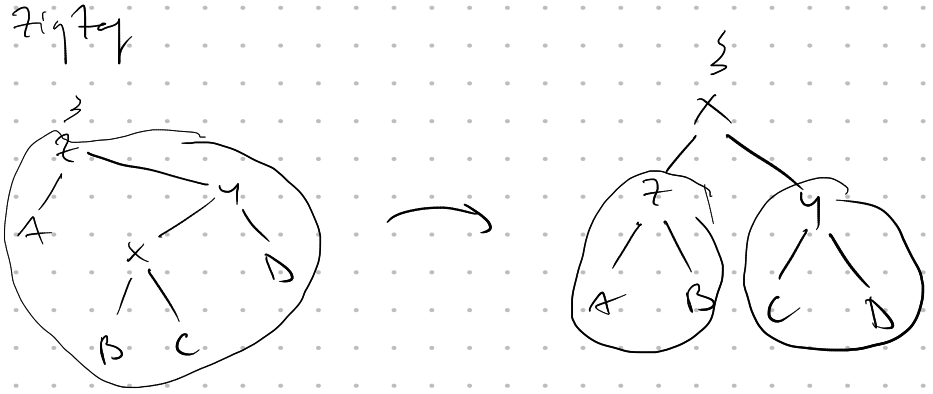
$A^2 \ge (B+C)^2 \;/\; -4BC$

$A^2 - 4BC \ge (B-C)^2 \ge 0$

$A^2 \ge 4BC \;/\; \log_2$

$2\log A \ge 2 + \log B + \log C$

zig-zig



Obs. $s(z) \geq s(x) + s(y)$

some trick:

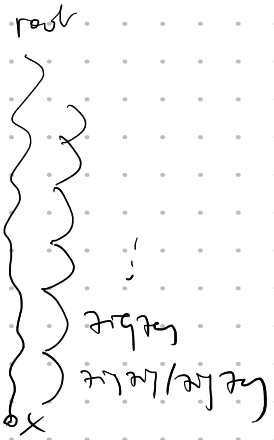
$$2r(z) \geq 2 + r(x) + r(y)$$

$$\Delta\phi = r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

$$\boxed{\Delta\phi \leq 2(r'(x) - r(x)) - 2}$$

in zig-zig / zig-zag: $\Delta\phi \leq 4(r'(x) - r(x)) - 2$

in zig: $\Delta\phi \leq 2(r'(x) - r(x))$



Add up $\Delta\phi$ due to all zig-zig, zig-zag, zig

(telescopes along search path)

last zig

$$\Delta\phi \leq 4(r(\text{root}) - r(x)) - \text{length of search path} + 1$$

$$\leq 4\log n - c_i + 1$$

$$a_i = c_i + \Delta\phi \leq c_i + 4\log n - c_i + 1 \in O(\log n)$$

□

(Proof of Thm 1.)

Thm 2. Let T be an arbitrary BST over $\{1, \dots, n\}$.
Serving R in a splay tree (with arbitrary i.t.)
has total cost $O\left(\sum_{i=1}^{m} d_T(r_i) + m + n \log n\right)$

Proof

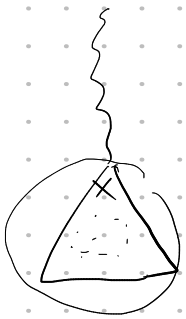
Adapt proof of thm 1.

assign weight $w(x)$ to node x ($w(x) = 1$ would give previous result)

$w(x) > 0$

$s(x) = \sum_{y \text{ in subtree}(x)} w(y)$ → Sum of weights in subtree

$r(x) = \log s(x)$



by same proof:

$$a_i \le 4\left(r(\text{root}) - r(x)\right) \le^{*}$$

$r(\text{root}) = \log\left(\sum_{x \in T} w(x)\right)$ ← denote sum of all weights by W

$r(x) = \log s(x) \ge \log w(x)$

$$\le^{*} 4\left(\log W - \log w(x)\right)$$

$$= 4 \log \frac{W}{w(x)}$$

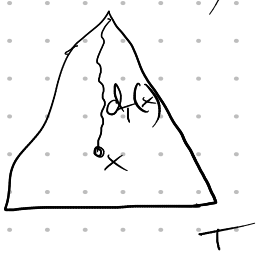
(if $w = 1$, recover Thm 1)

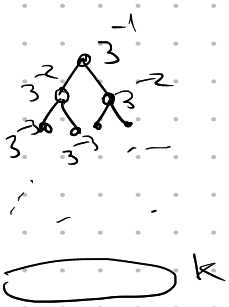
How to set weights $w(x)$?

With different choice of weight we can prove Thm 4, Thm 5

Want: $O\left(\sum_{i=1} d_T(r_i) + m + n \log n\right)$

Idea: set $w(x) = 3^{-d_T(x)}$ (depth in reference tree T)





amortized cost $a_i = O\left(\log \frac{W}{w(x)}\right)$ $(x = r_i)$

$= O\left(\log 3^{d_T(x)}\right)$

$a_i = O(d_T(x))$. (Thm 2)

at level k: $2^k \cdot 3^{-k} = \left(\frac{2}{3}\right)^k$

$W \le \sum_{k=0}^{n} \left(\frac{2}{3}\right)^k \le 3 \in O(1)$