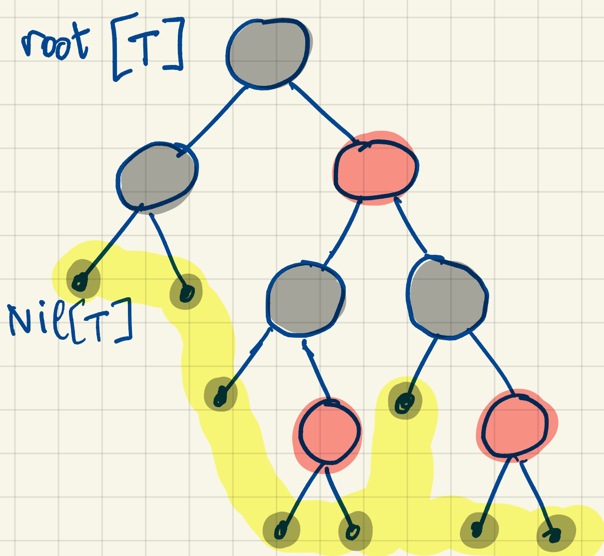
Red-black trees & skip Lists

Red-black Tree: Binary Search tree with color / node which is either red or black

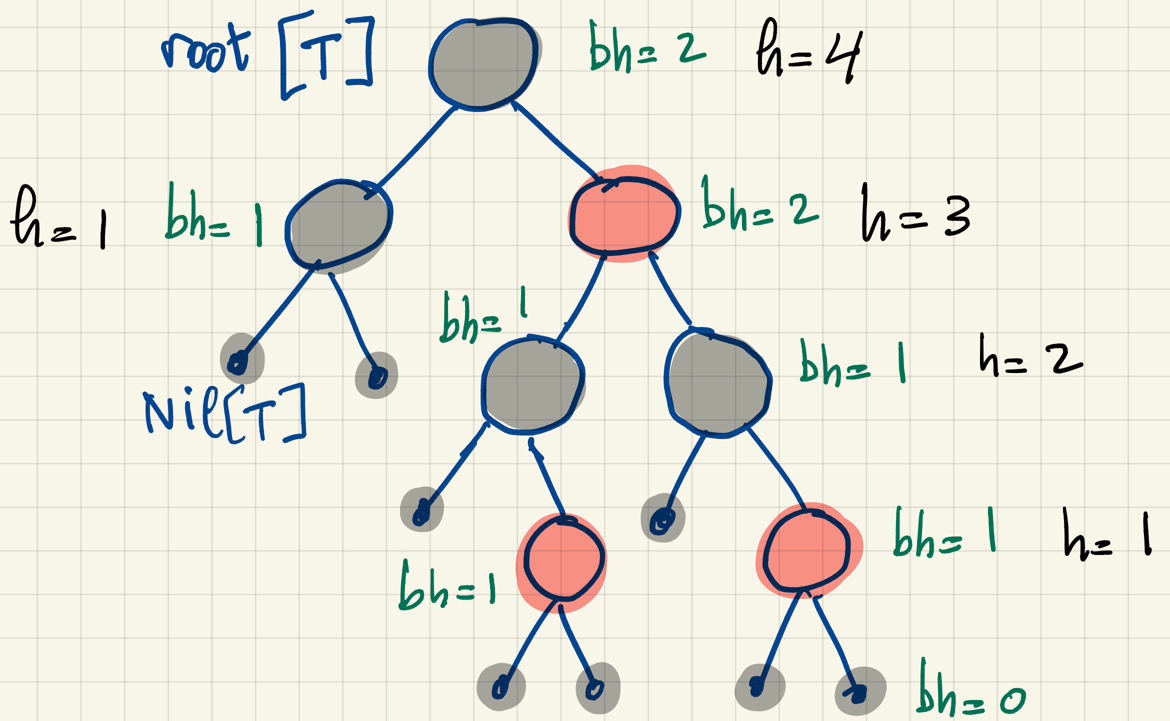
Assume : root[T] is the root
Nil[T] is a single sentinal for all leaves
p[root[T]] = Nil[T]



Red-Black tree properties

1) Root is black
2) Nil[T] is black
3) The children of red node are black
4) All paths from a node x to Nil[T] contain same number of black nodes.

Definition : The black height of a node x, $bh(x)$, is the number of black nodes on a path from x to Nil[T], not counting x itself.



So what's good about red-black tree? The height of tree is $O(\log n)$ where n = # internal nodes (not Nil[T]) claim 1: $bh(x) \geq h(x)/2$. [Property 3] $\leq \frac{h}{2}$ nodes on the path are red. So $\geq \frac{h}{2}$ are black.

claim 2: The subtree rooted at x has at least $2^{bh(x)}-1$ internal nodes.

proof by induction:

Base case. x is Nil[T], then $bh(x)=0$ and $2^0-1=0$ is the number of internal nodes in x's subtree

Inductive step: Consider left[x] and right[x]. Each has black height at least $bh(x)-1$. By inductive hypothesis, x's subtree has at least $1+2(2^{bh(x)-1}-1)$ internal nodes which is $2^{bh(x)}-1$.

Finally, the two claims show that $h(x) \leq 2bh(x) \leq 2\lg(n_x+1)$ where $n_x =$ # internal nodes in x's subtree.

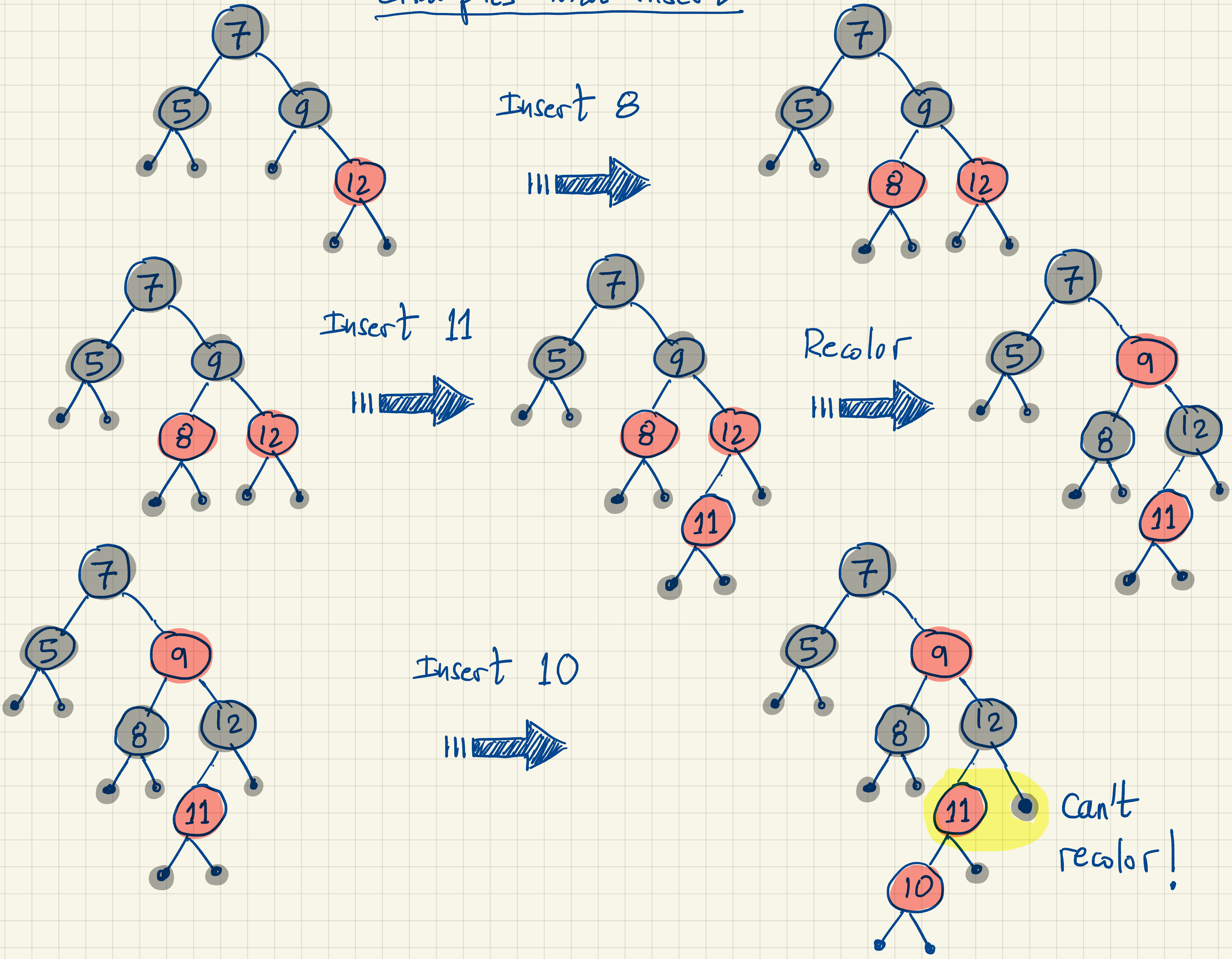
Operations on Red-black tree:

All operation that don't modify the tree run as before in $O(\log n)$ time.

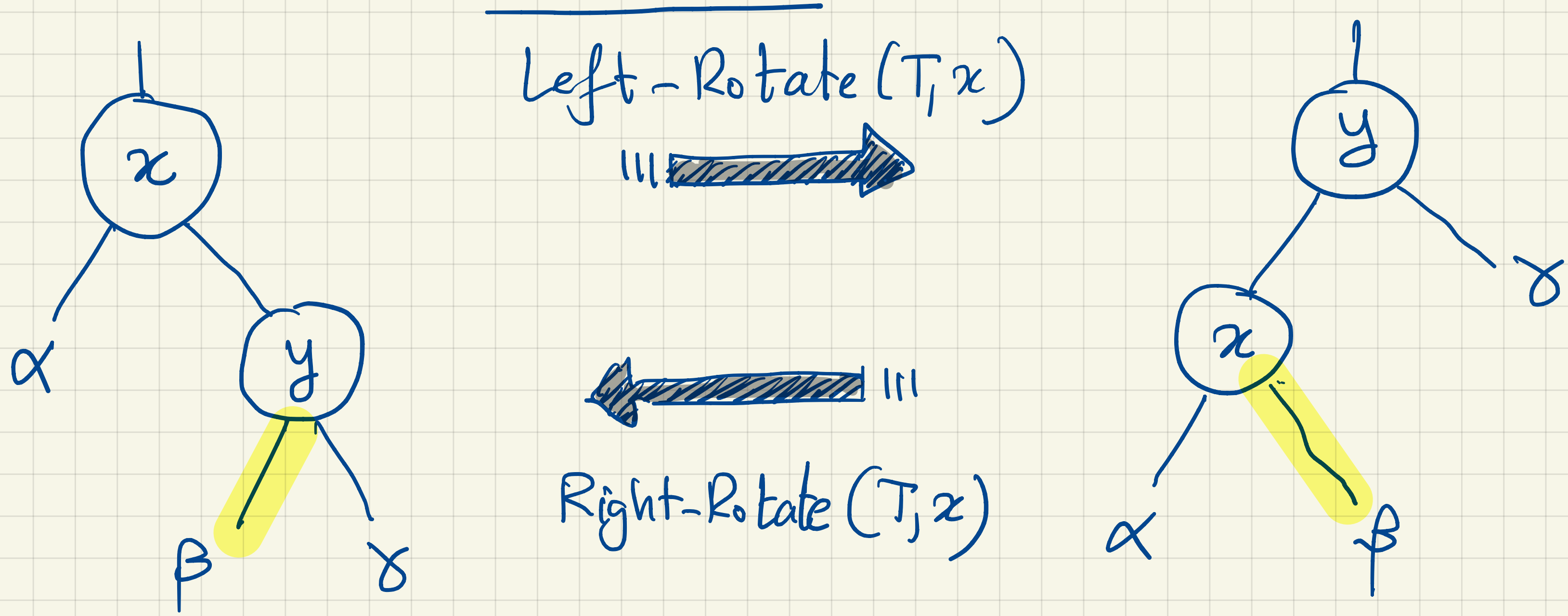
- Minimum
- Maximum
- Successor
- Predecessor
- Search

Problems to take care of: Insert & Delete.

Examples with Insert:



Rotations



- Rotations can be done in $O(1)$ by simple update of pointers.

- Do not affect the Binary Search Tree Property.

Left-Rotate (T, x)

$y \leftarrow right[x]$ $(\neq Nil[T])$

$right[x] \leftarrow left[y]$ (move β to x)

if $left[y] \neq Nil[T]$

then $p[left[y]] \leftarrow x$ (update parent of β)

$p[y] \leftarrow p[x]$ (update parent of y)

if $p[x] = Nil[T]$ (make y root)

then $root[T] \leftarrow y$

else if $x = left[p[x]]$

then $left[p[x]] \leftarrow y$

else $right[p[x]] \leftarrow y$

$left[y] \leftarrow x$

$p[x] \leftarrow y$

(put x on y's left)

Assumes

$right[x] \neq Nil[T]$

$p[root[T]] = Nil[T]$

How to Insert (T, z)

- Newly inserted node z is Red. This will break properties if

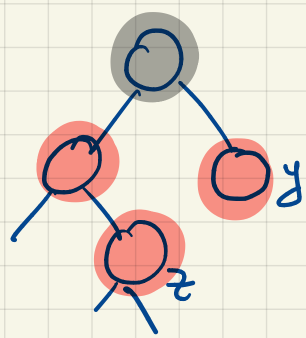
  1) z is root

  2) $p[z]$ is Red.

- Insert as usual, then call Insert_fixup(z)

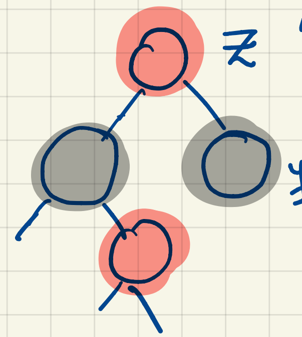
Three Cases

Case 1



"uncle"

Uncle is Red

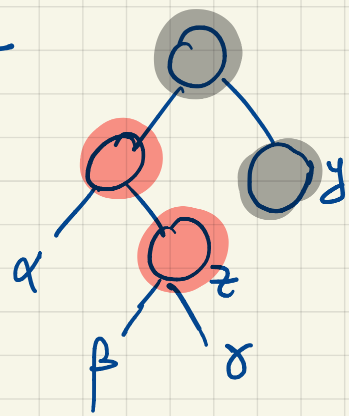


Now this is z

"uncle"

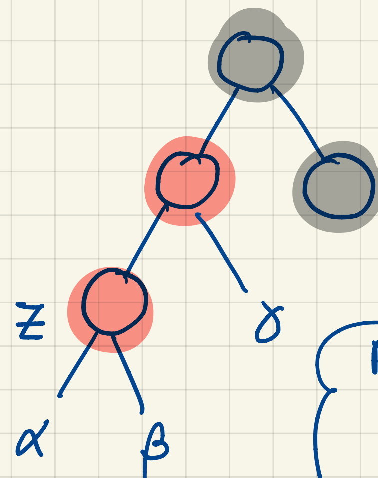
"fix" recursively

Case 2



"uncle"

Uncle is black

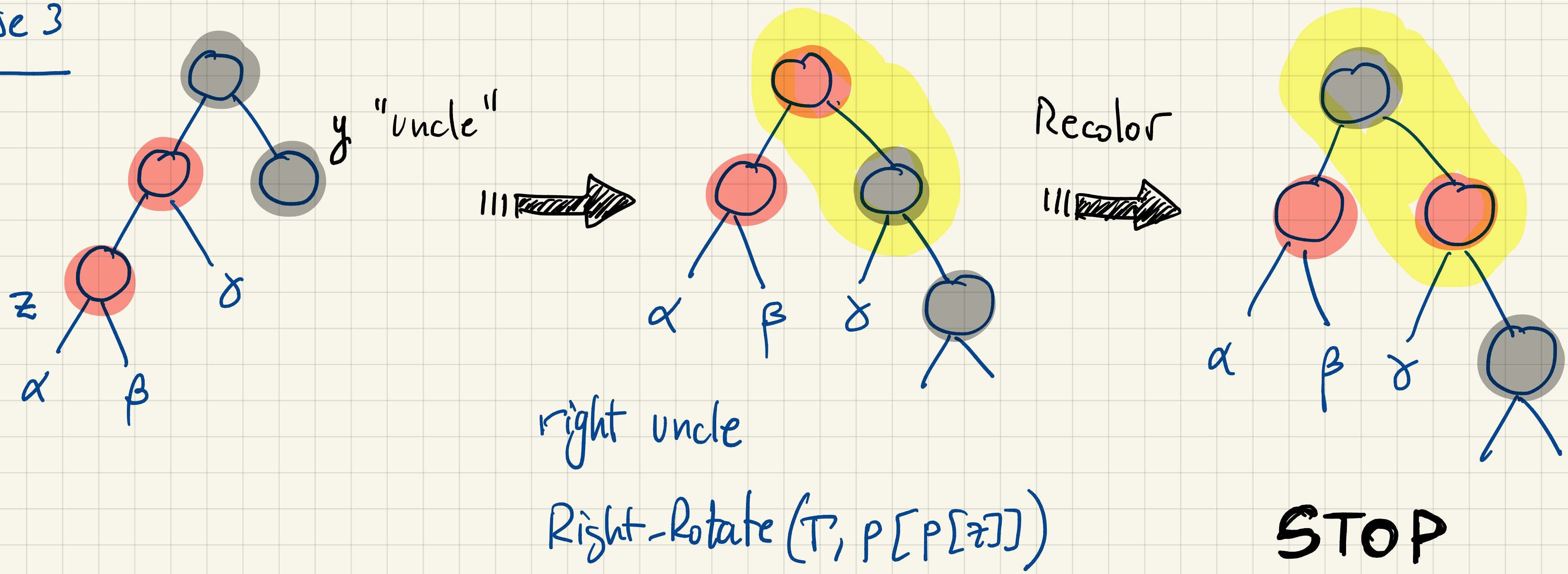


right uncle

left-Rotate(T, p(z))

Case 3

Case 3



Case 1 propagates up the tree $\Rightarrow$ $O(\lg n)$ time.

Case 2 and Case 3 are terminal $\Rightarrow$ $O(1)$ rotations.

Deletion, more complicated but can be handled similarly.

INsert-Fixup (z)

$\triangleright$ color of z is Red

while color[p[z]] = Red

do if p[z] = left[p[p[z]]]

then $y \leftarrow$ right[p[p[z]] $\triangleright$ right uncle

if color[y] = Red

then Case 1 $\triangleright$ recolor & $z \leftarrow$ p[p[z]]

else if z = right[p[z]] $\triangleright$ Case 2

then $z \leftarrow$ p[z]

Left-Rotate (z)

$\triangleright$ Case 3

color[p[z]] $\leftarrow$ Black

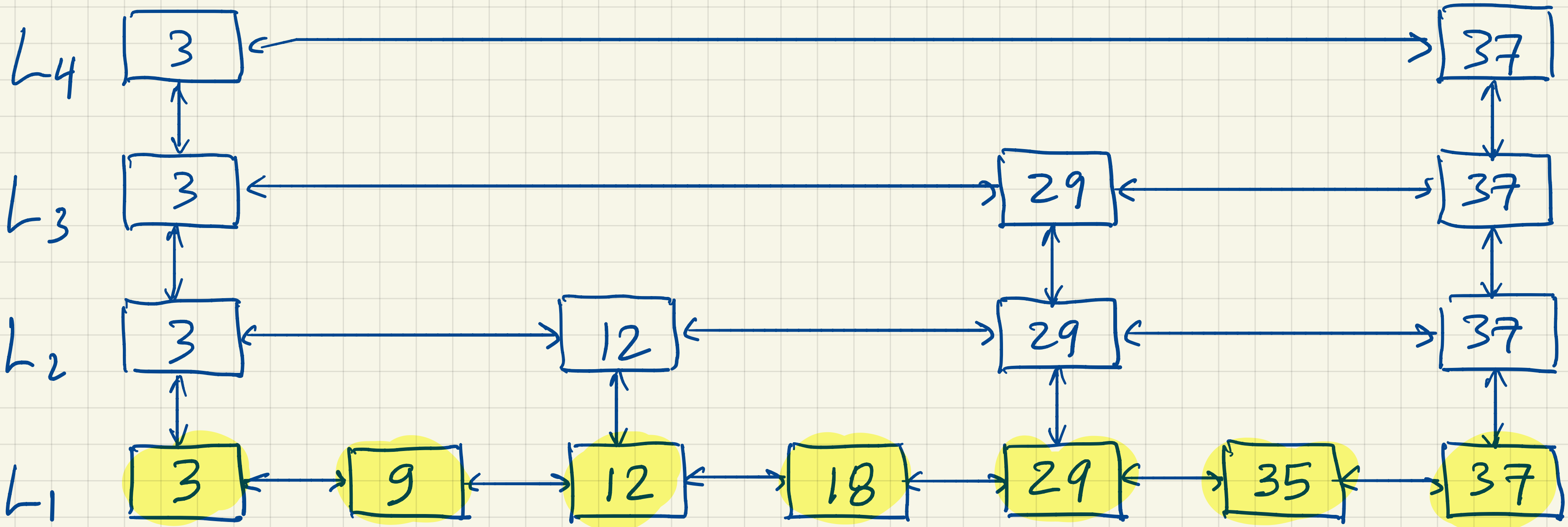
color[p[p[z]] $\leftarrow$ Red

Right-Rotate (p[p[z]])

else (symmetric)

color[root[T]] $\leftarrow$ Black

Skip List : Sorted doubly linked list with levels that skip. (See below)



- Min, Max, Successor, Predecessor can be done in $O(1)$ time (Assume we also have a pointer to the end for Max)
- Delete $O(1)$ given what to delete
- Insert is $O(1)$ + Search time.

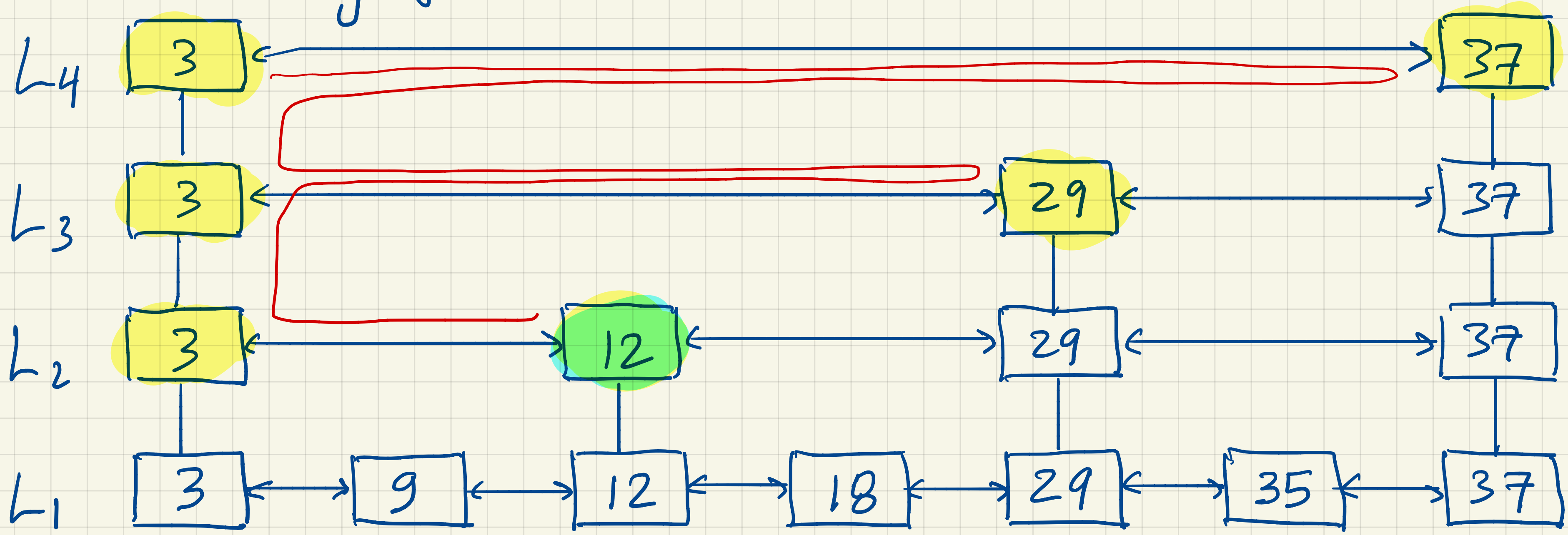
How to search? Start with highest level.

- In level i, find j such that

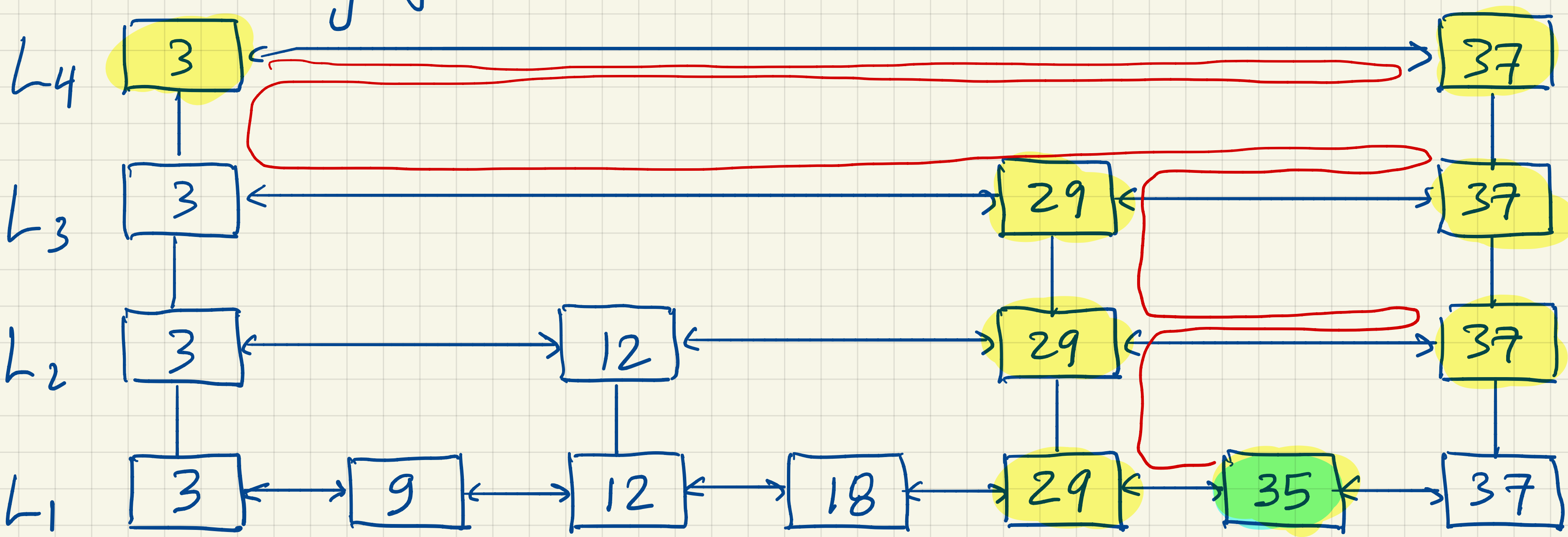
$$x \in [L_i(j), L_i(j+1)]$$

- If x is $L_i(j)$ or $L_i(j+1)$, then done.
- Otherwise follow pointer from $L_i(j)$ to level$_{i-1}$.
- If not found in Level 1, then it's not there

Searching for 12

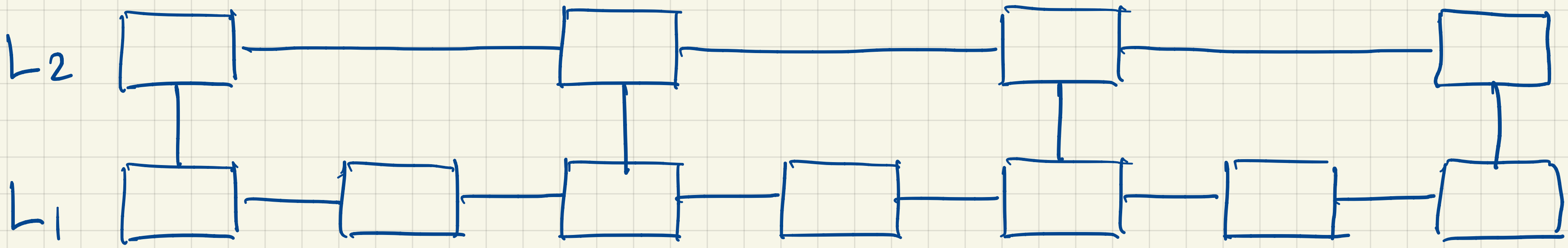


Searching for 35



How to make Search $O(\log n)$ time instead of $O(n)$ time?

Let's look at case of two levels



Assume elements in L_2 are uniformly distributed

Then Search time is

$$|L_2| + \frac{|L_1|}{|L_2|} = n$$

Approximately, this is minimized when $|L_2| = \frac{|L_1|}{|L_2|}$

So $|L_2| = \sqrt{n}$ and Search time $\approx 2\sqrt{n} = 2n^{1/2}$

- In general, for r levels, the search will take approximately $r n^{1/r}$ time.
- What if we choose $r \approx \lg n$, then

$$\lg n \cdot n^{1/\lg n}$$

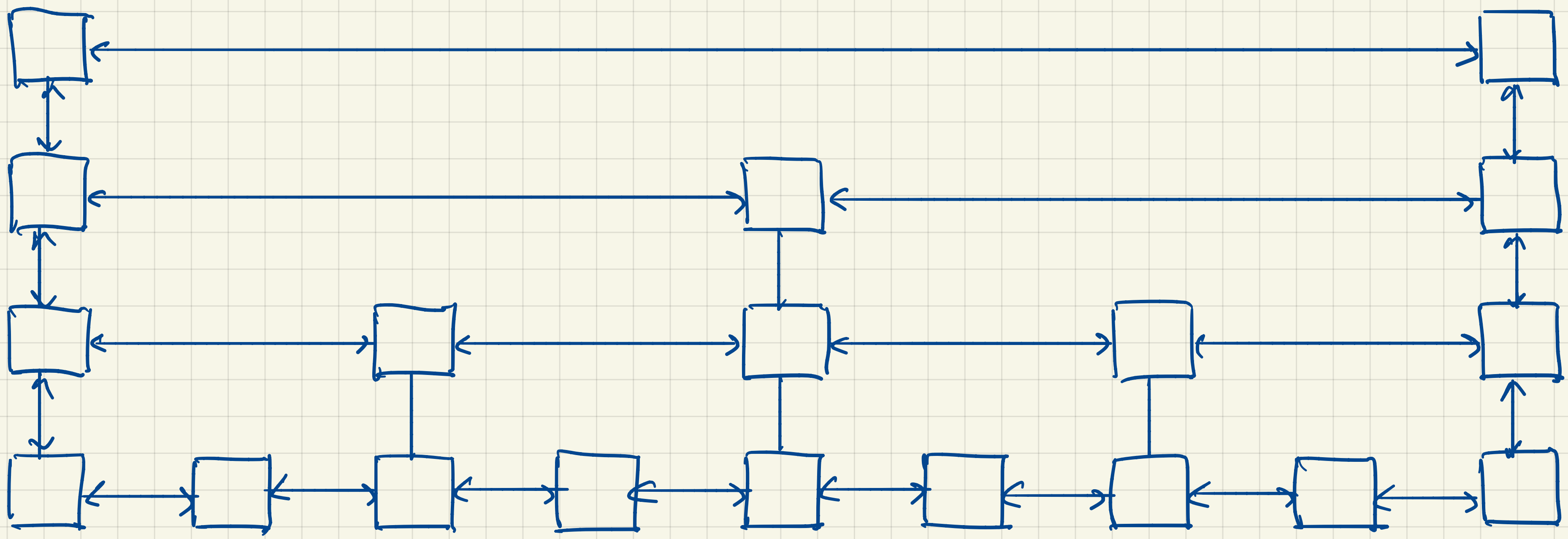
but $n^{1/\lg n} = 2$, so we have

$$\approx 2 \lg n \text{ time.}$$

- Since $\frac{|L_1|}{|L_2|} + \frac{|L_2|}{|L_3|} + \dots + \frac{|L_{r-1}|}{|L_r|} + |L_r| = 2 \lg n$

and $r = \lg n$, the ratio $\frac{|L_i|}{|L_{i+1}|} \approx 2$

Next level has $\approx \frac{1}{2}$ the elements.



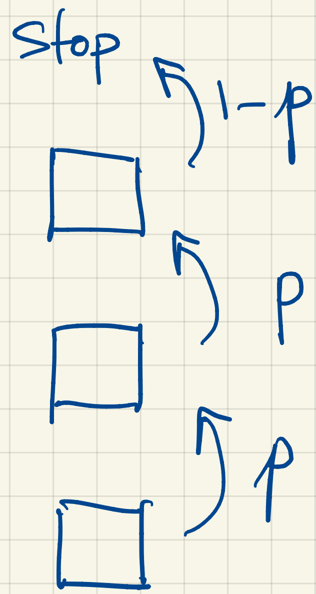
But how do we maintain this with insertions & deletions?

Deletion: Easy, when deleting an element, delete it all the way.

Insertion: Don't insist on "ideal" structure.

When inserting an element x,

- with prob. p "promote" it to the next level
- repeat, until x is not promoted.

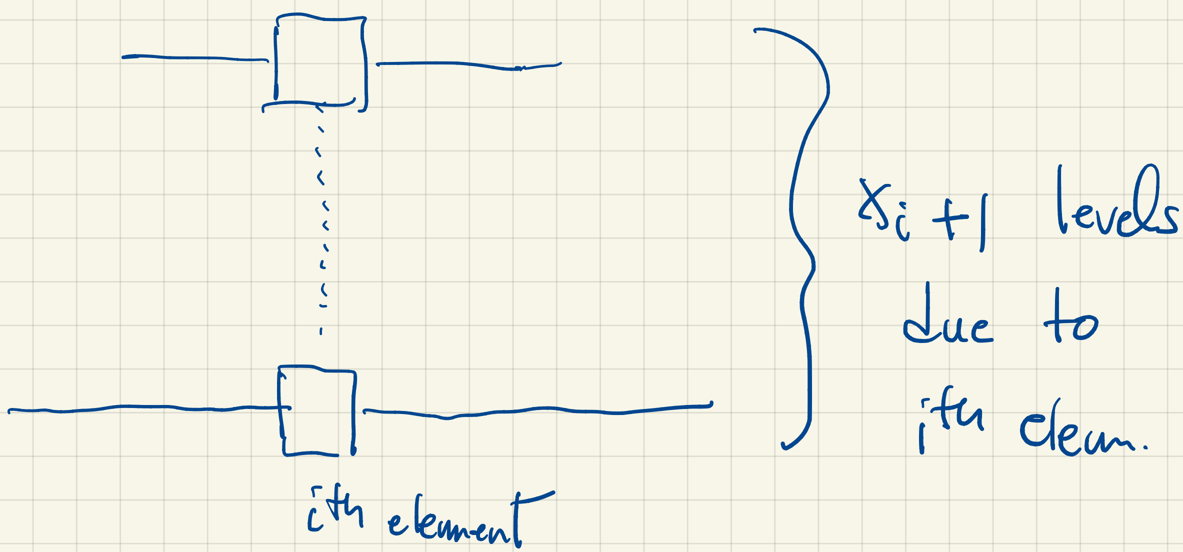


e.g. choose $p \approx \frac{1}{2}$

$r = \#$ levels

$x_i = \#$ times i^{th} elem was promoted.

$$r = 1 + \max_{i=1..n} x_i = \max_{i=1..n} [1 + x_i]$$



let $Z = \max_{i=1..n} x_i$

Show: $P(Z \geq t) \leq n p^t$

$$P(Z \geq t) \leq P(\underline{X_1 \geq} t) + P(\underline{X_2 \geq} t) + \dots + P(\underline{X_n \geq} t)$$

$$P(E_1 \text{ OR } E_2) \leq P(E_1) + P(E_2)$$

$$Z \geq t \iff (X_1 \geq t) \text{ OR } (X_2 \geq t) \text{ OR } \dots (X_n \geq t)$$

$$P(Z \geq t) = P(X_1 \geq t \text{ OR } X_2 \geq t \text{ OR } \dots X_n \geq t)$$

$$\leq \underset{pt}{\underline{P(X_1 \geq t)}} + \underset{pt}{P(X_2 \geq t)} + \dots + \underset{pt}{P(X_n \geq t)}$$

<u>Analysis</u>: let x_i = # times we promote i^{th} elem.

$$P\left(\max_{i=1..n} x_i \geq t\right) \leq n P(x_i \geq t) \quad \text{[union bound]}$$

$$P(x_i = k) = p^k(1-p) \quad k \geq 0$$

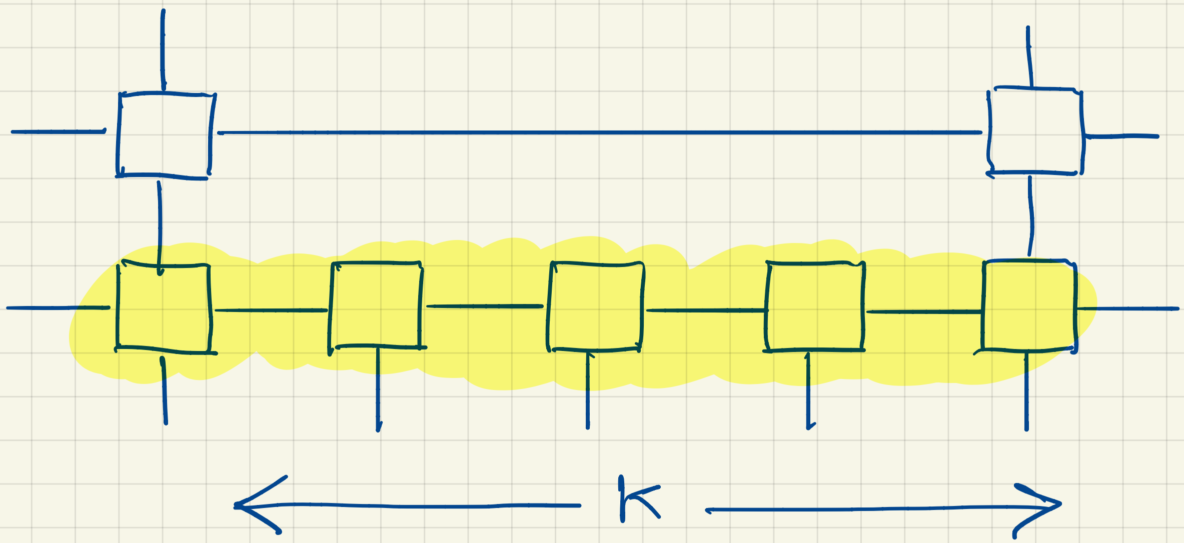
So $P(x_i \geq t) = (1-p)(p^t + p^{t+1} + \cdots) = (1-p)\frac{p^t}{1-p} = p^t$

let $t = \alpha \lg n$ and $p = \frac{1}{2}$

$$P\left(\max_{i=1..n} x_i \geq \alpha \lg n\right) \leq \frac{n}{n^\alpha} = \frac{1}{n^{\alpha-1}}$$

So with high prob. $r = O(\lg n)$

But how much do we search in each level?



$P(Y=K) = (1-p)^{K-1} p \quad (K \geq 1)$. So Y is a geometric random variable with $O(1)$ Expectation.

$E[Y] = 1/p$