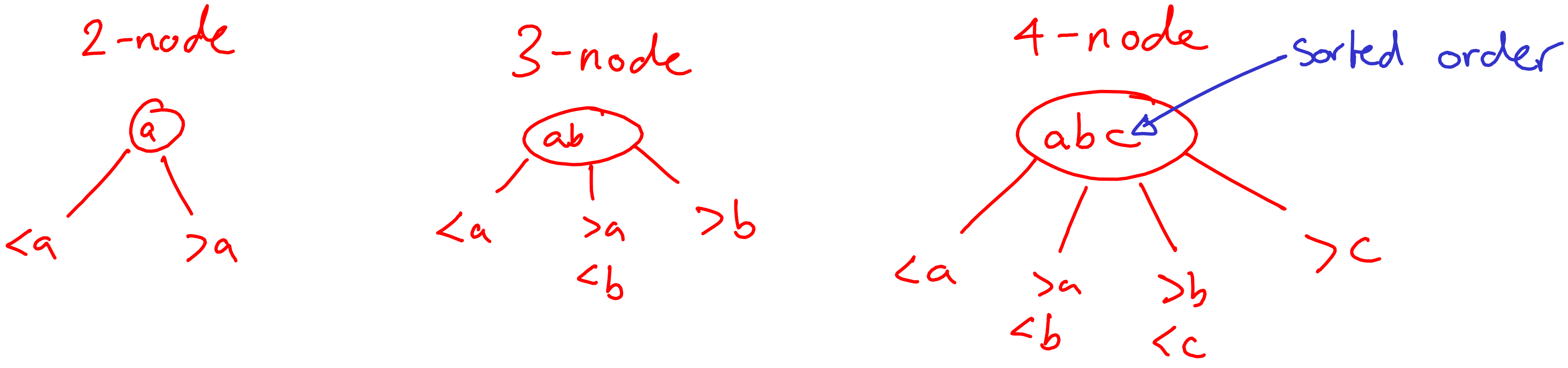
Red-Black Trees

(You will need to make notes on
this lecture ...)

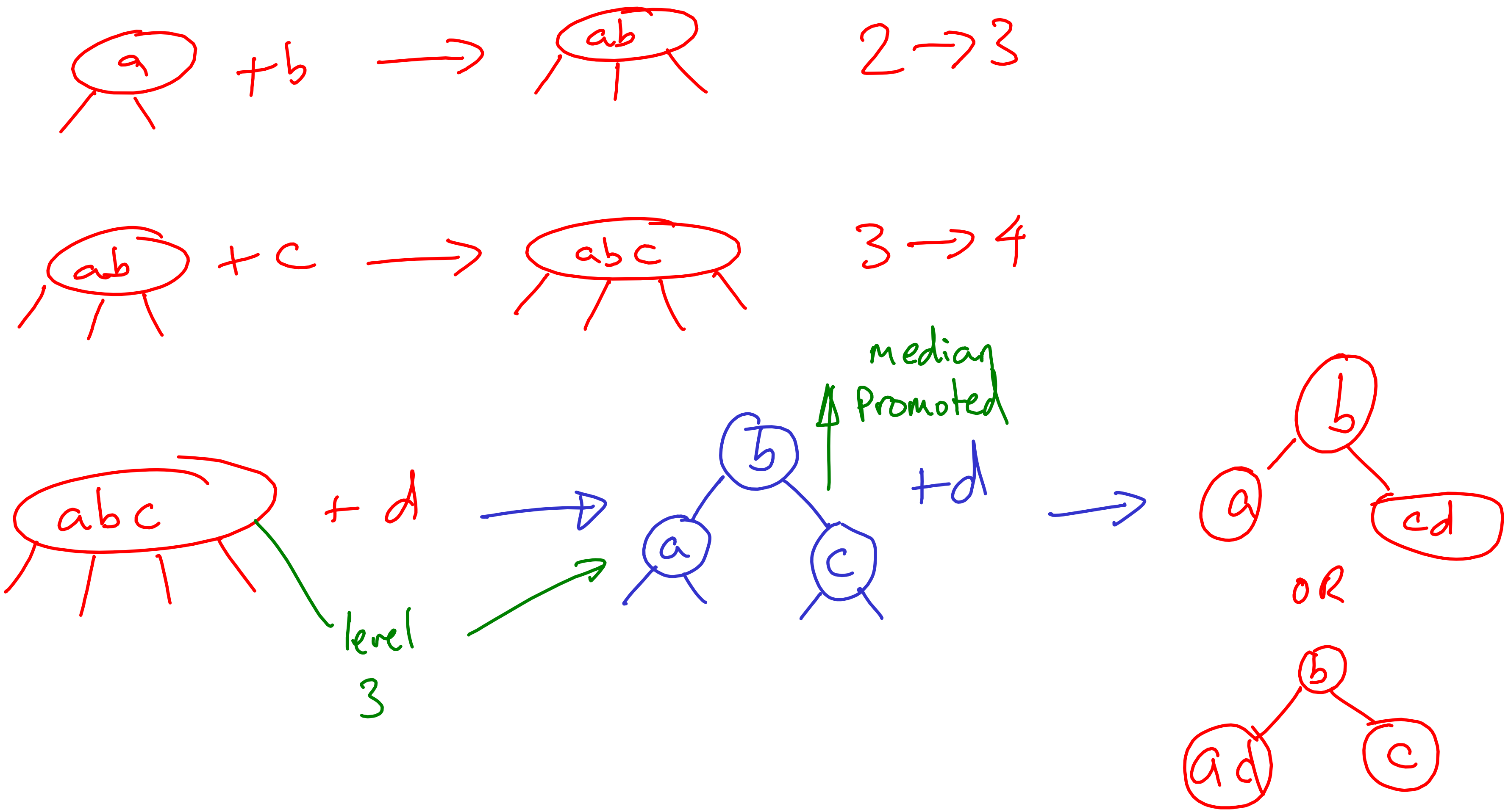
2-3-4 Trees



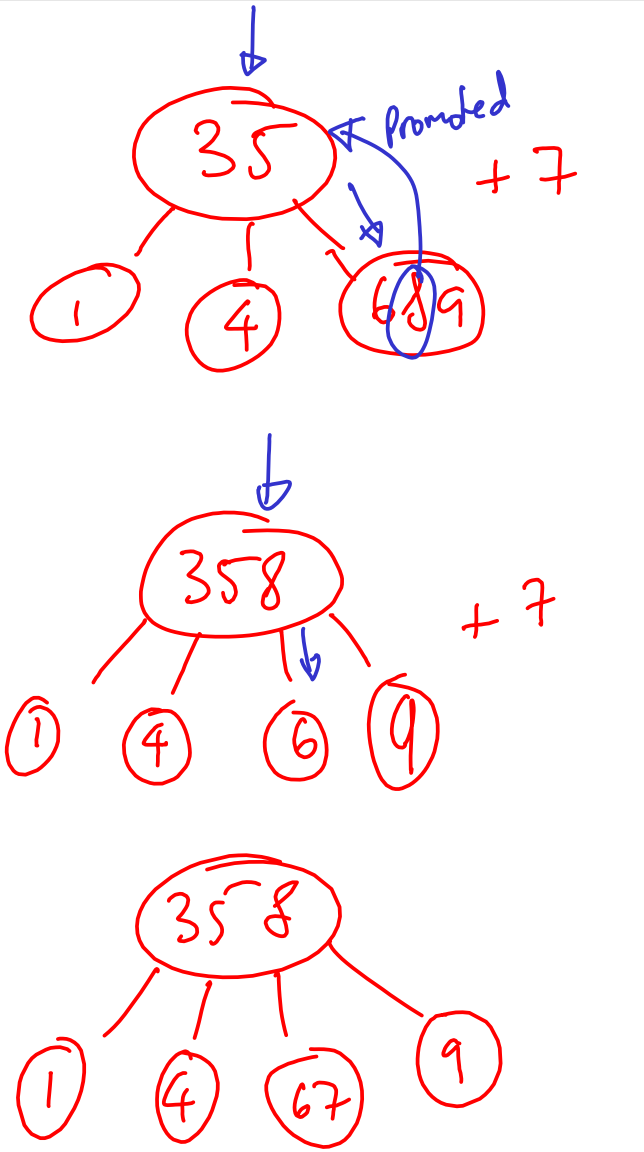
$$N_{keys} = N_{children} - 1$$

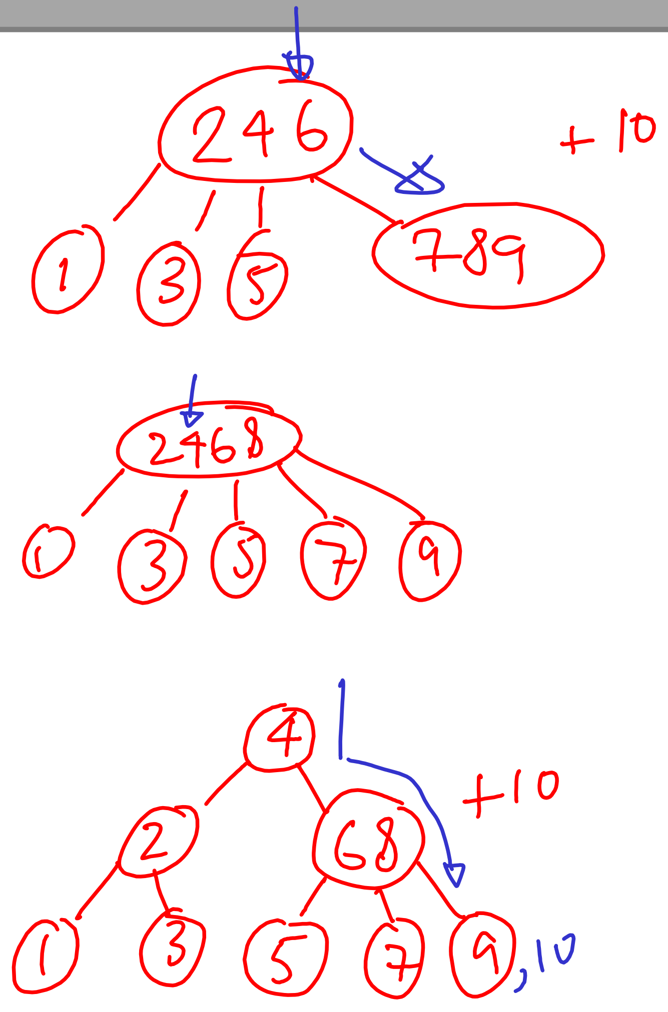
Insertion

Special rules

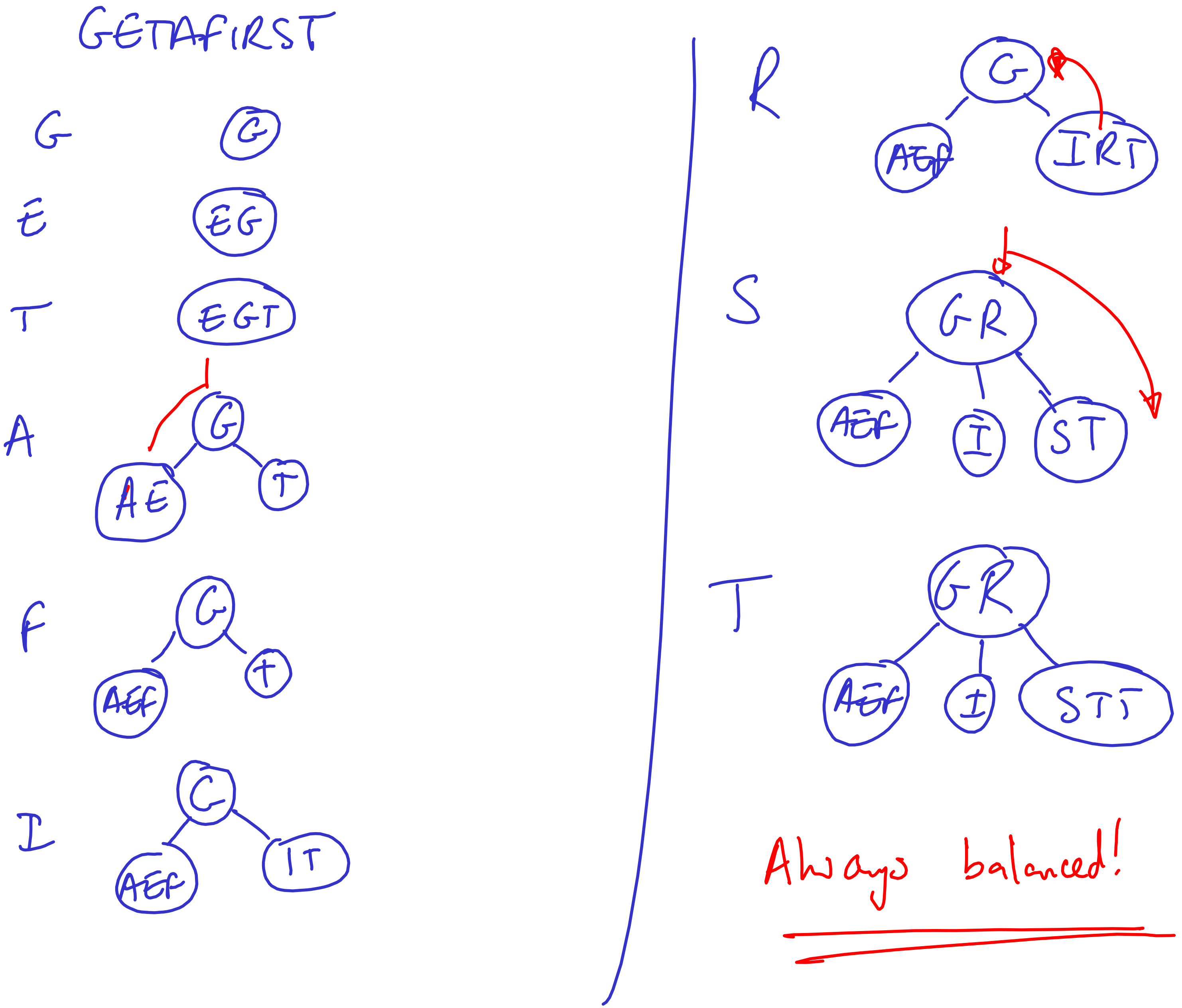


Example



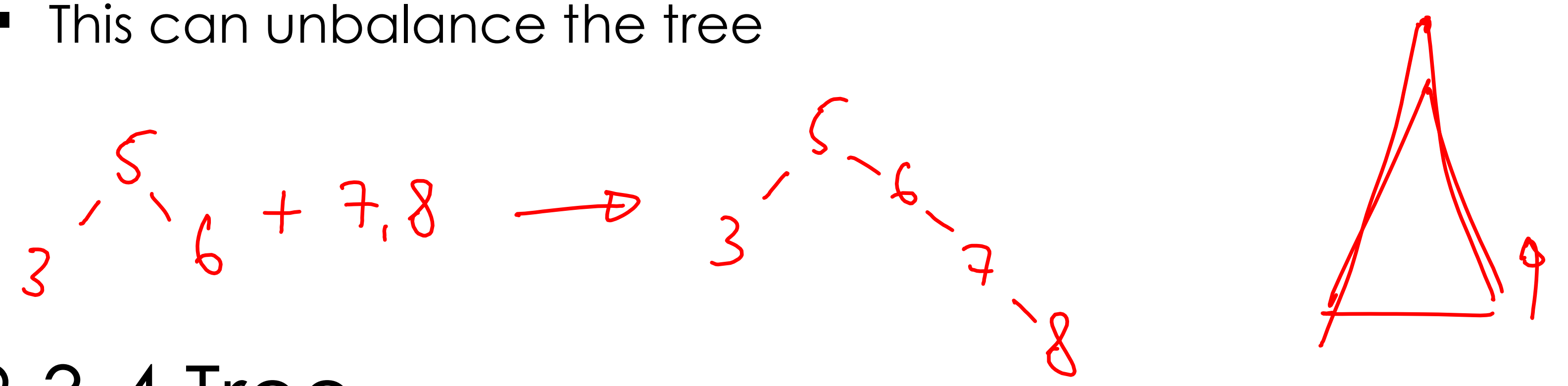


Example

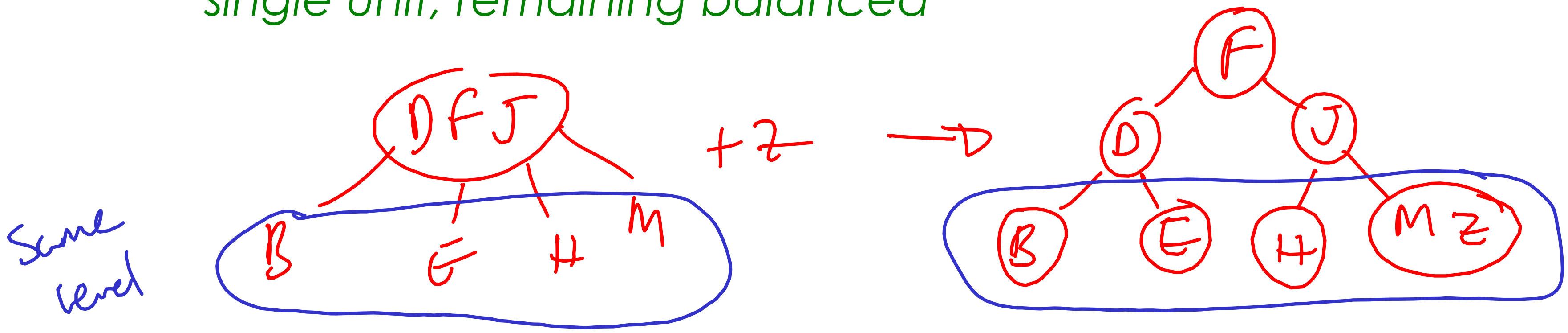

GETAFIRST
G
G
E
EG
T
EGT
A
G
AE
T
F
G
AEF
T
I
G
AEF
IT
R
G
AEF
IRT
S
GR
AEF
I
ST
T
GR
AEF
I
STT
Always balanced!

Why 2-3-4 Trees Remain Balanced

- Binary Tree
  - Additions add nodes to the bottom
  - This can unbalance the tree



- 2-3-4 Tree
  - Additions only push nodes *up*
  - *So the base of the tree is effectively pushed down as a single unit, remaining balanced*

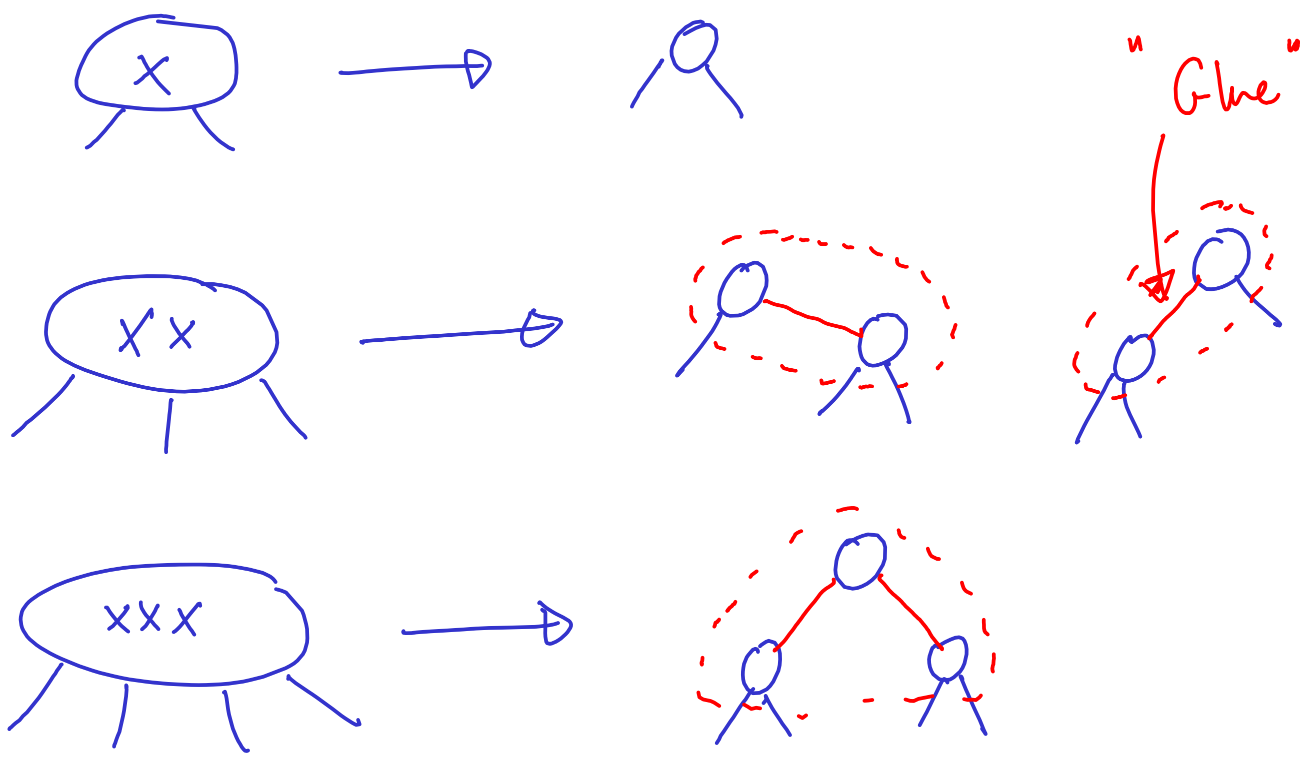


Difficulties with 2-3-4 Trees

- The main problem is that those different node sizes have different storage requirements
  - Converting a node from one type to another Is going to be costly in a computer
  - You could just implement 4-nodes and not use the spare links when you want a 2- or 3-node.
    - Can be very wasteful

Binary tree implementation

- Let's keep our beloved binary trees and try to 'hack' 2-3-4 capabilities into it.
- Firstly we have be able to represent the different node types:



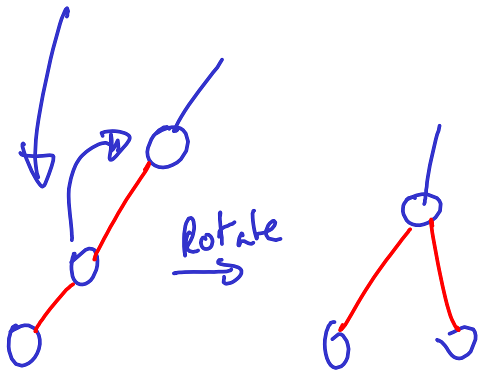
Insertion Rules

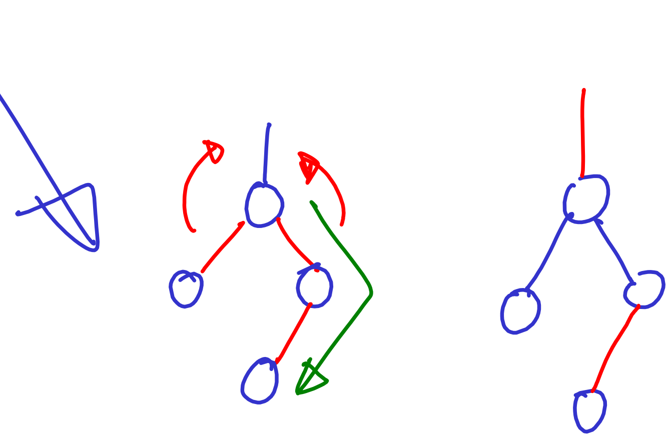
Insert as per BST

Colour new link red

```
if (two reds in a row) {
   if (4-node equivalent) promote to fix
   else rotate to fix
}
```

"Red violation"





Example 1

GETAFIRST

G

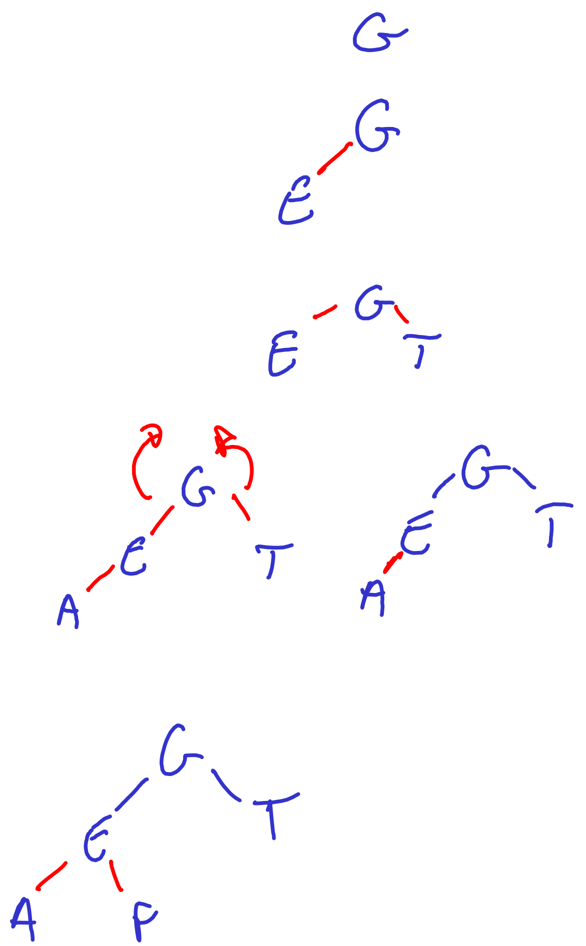
E

T

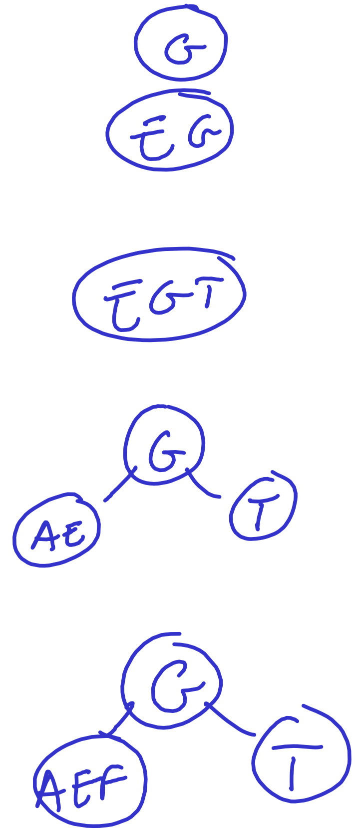
A

F

RB

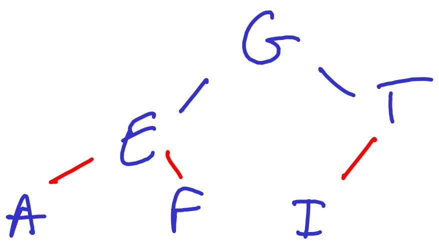


2-3-4

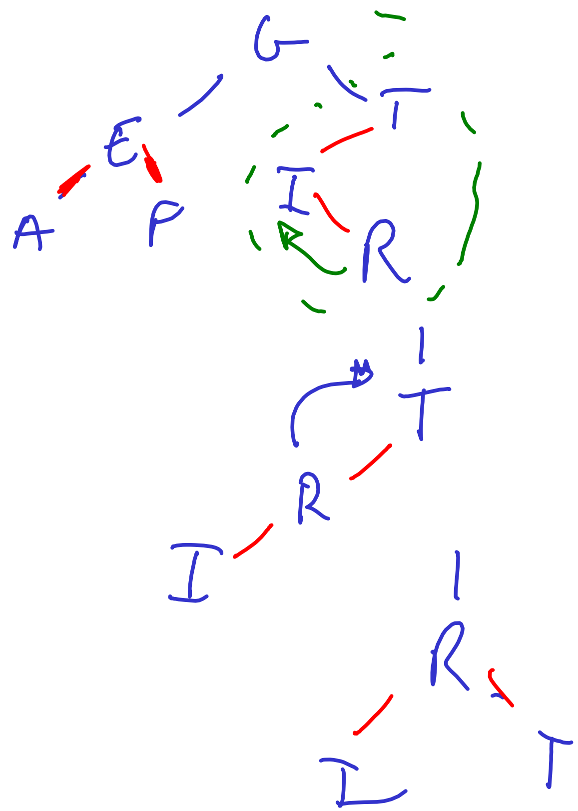


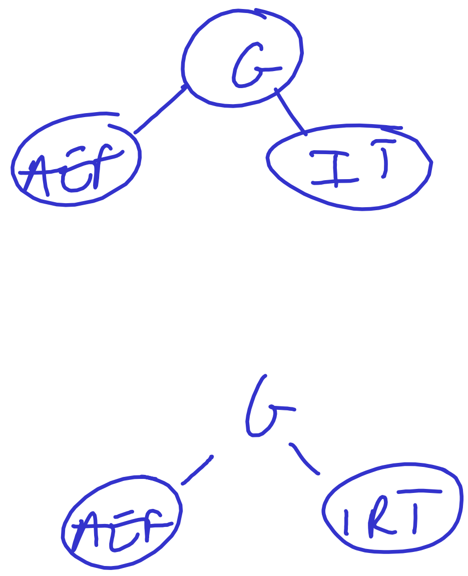
Example 1

I



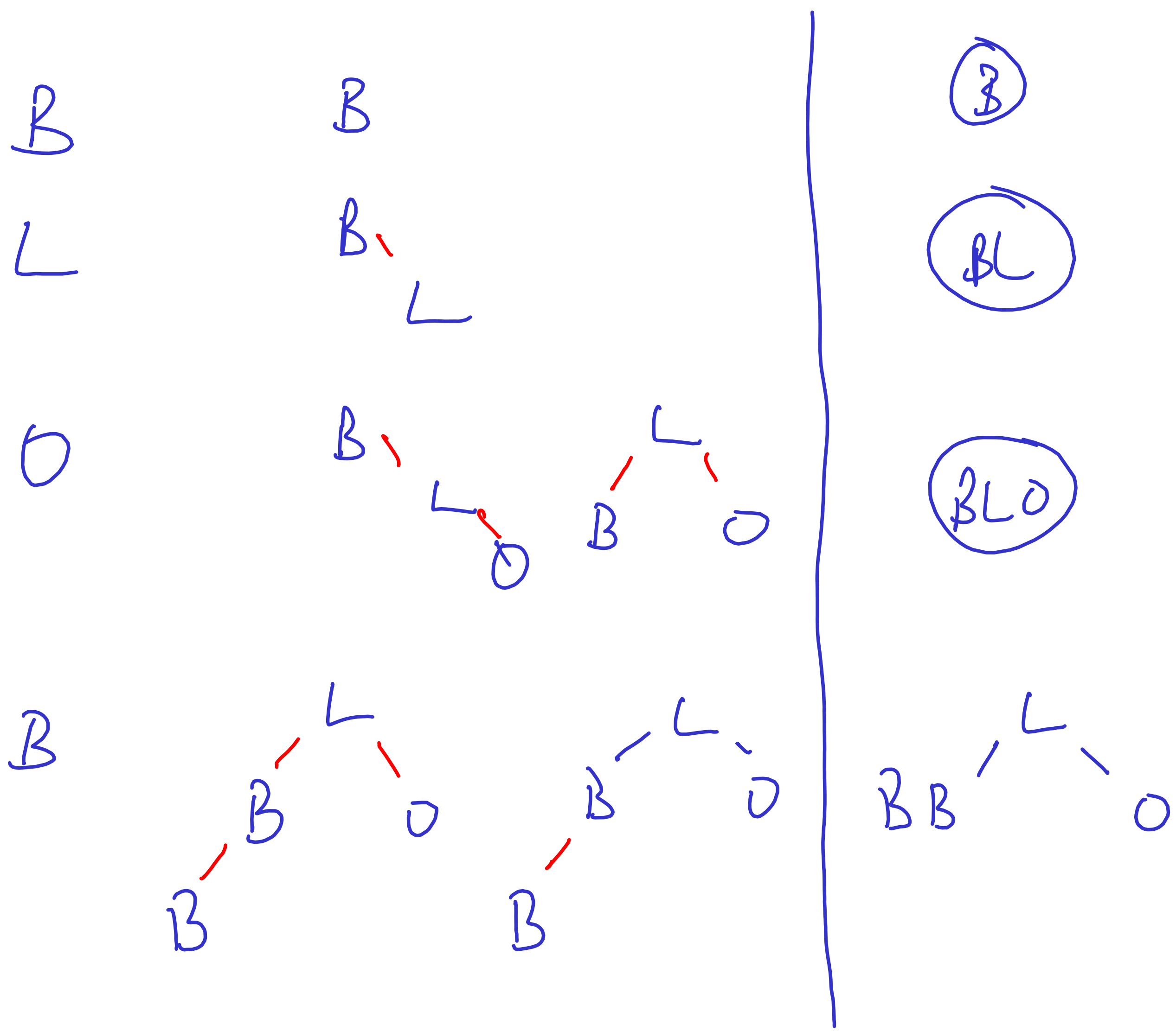
R





Example 2

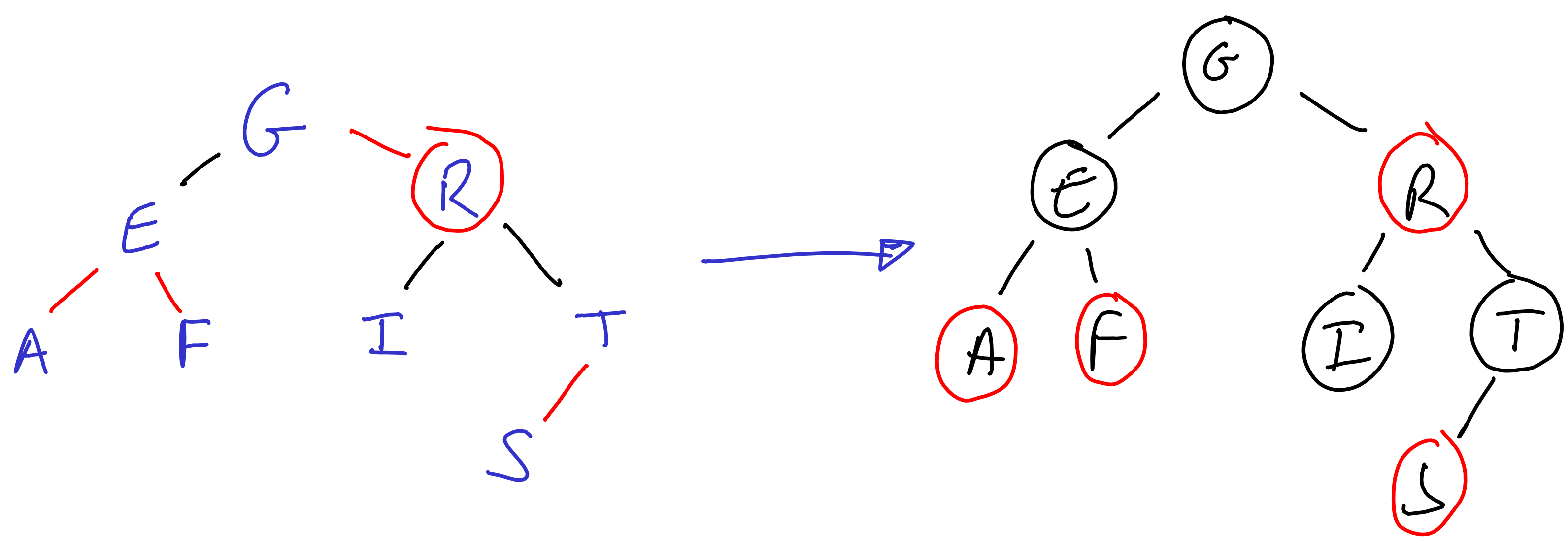
BLOBFISH



Example 2

The 'Normal' View

- Normally the links aren't represented as objects with properties in our code, just the nodes
  - So we 'colour' the nodes according to the incoming (parent) link
  - This is the usual way to view a "red-black tree"



The Red-Black Properties

- Every node is red or black
- The root node is black
  - Because red nodes are linked to the ends of the red links we had. The root has no parent so cannot be on the end of any link
- If a node links to a NULL node, the NULL 'node' is black
  - Otherwise we'd have an incomplete 2-3-4 node

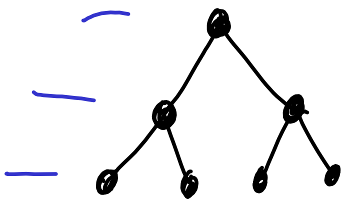
The Red-Black Properties

- Every red node has black children
  - None of the binary representations of 2-3-4 nodes requires two consecutive red links so black <u>must</u> follow red

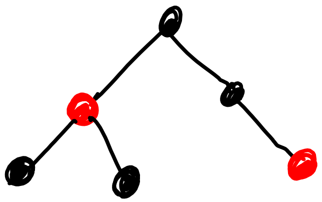
- Every path from the root to a leaf must visit the same number of black nodes
  - There is one black node for each 2-3-4 node and we know 2-3-4 trees are balanced so the black nodes represent the path through the balanced 2-3-4 tree.

Red-Black Analysis

- We know that whichever route we take from the root to a leaf, we meet the same number of black nodes
  - Since the 2-3-4 tree nodes equate to black nodes in the RB tree
  - Call this the black height of the tree, BH



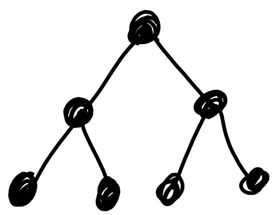
BH=3



BH=2

How many nodes?

- If the tree was purely black, we would have:



Always a full tree if purely black

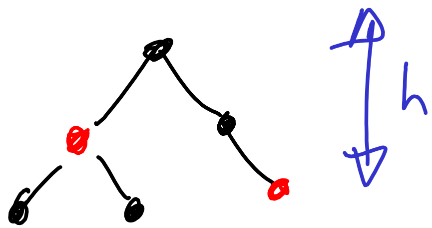
$$n = 2^{BH} - 1$$

- Adding in red nodes doesn't change the black height so we know

Total no. of nodes $= n \geq 2^{BH} - 1$

How many nodes?

- In the worst case, there is one red node for every black node



BH at least $h/2$

$$BH \geq \frac{h}{2}$$

So what is h?

$BH \geq \frac{h}{2}$ $\qquad n \geq 2^{BH} - 1$

$n \geq 2^{BH} - 1$

$n \geq 2^{\frac{h}{2}} - 1$

$n+1 \geq 2^{h/2}$

$\log_2(n+1) \geq \frac{h}{2}$

$h \leq 2\log_2(n+1)$

h is $O(\lg n)$

All BST operations were $\sim O(h)$

$\therefore \Rightarrow O(\lg n)$ for Red-Black tree

Red-Black Performance

	Average	Worst Case
Space	$O(n)$	$O(n)$
Insert	$O(\lg n)$	$O(\lg n)$
Delete	$O(\lg n)$	$O(\lg n)$
Search	$O(\lg n)$	$O(\lg n)$