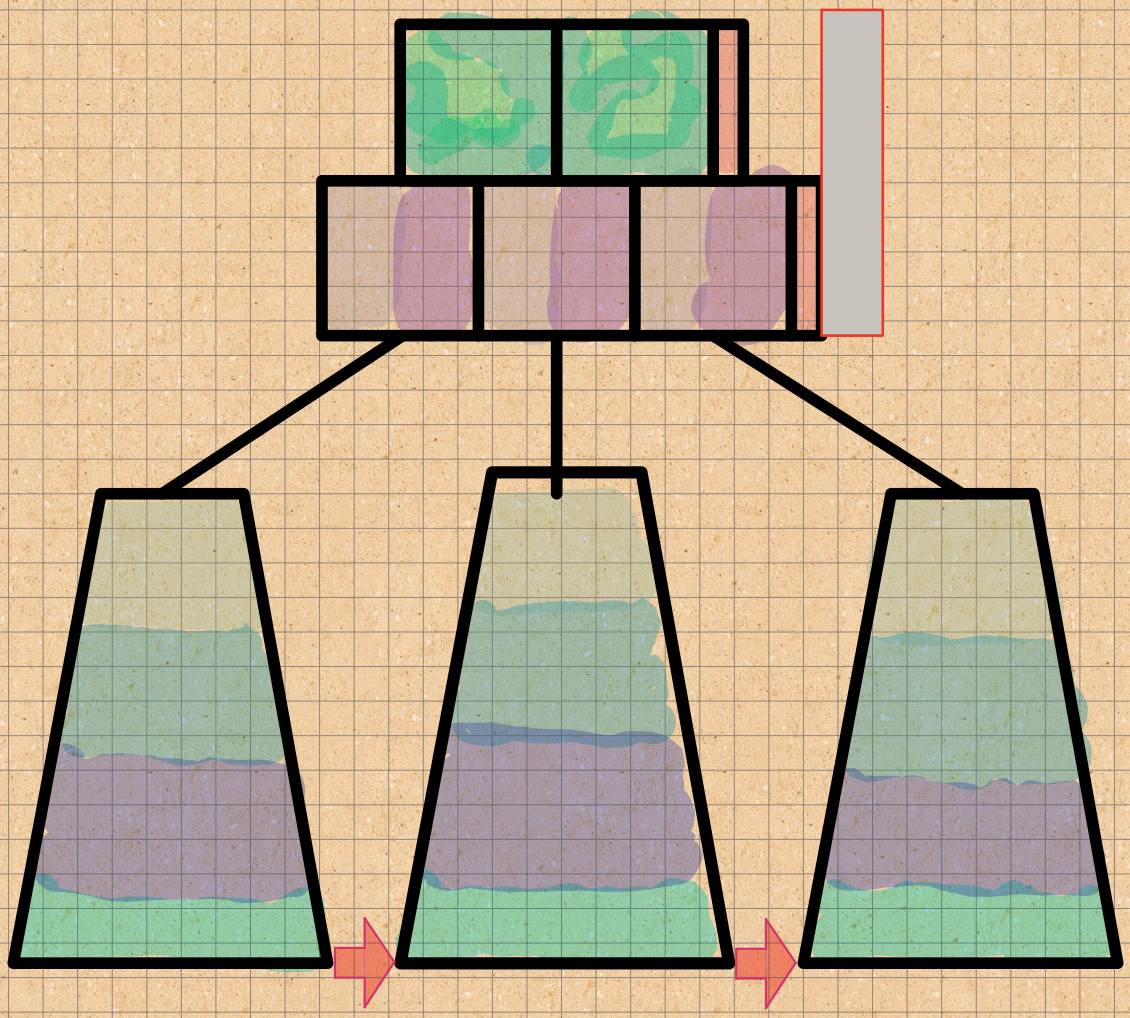
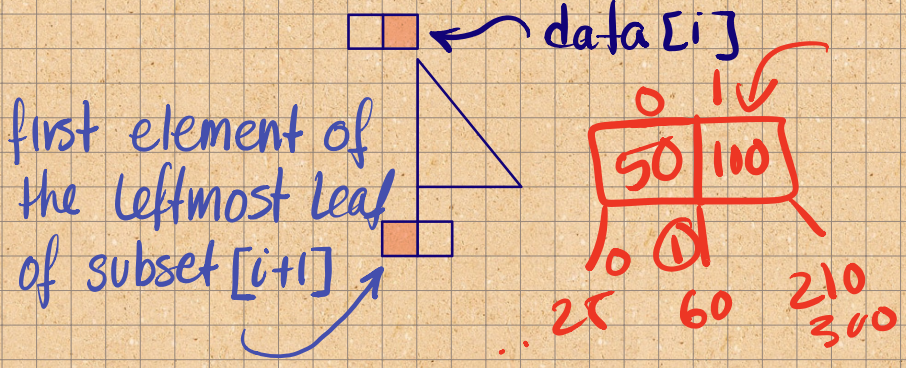
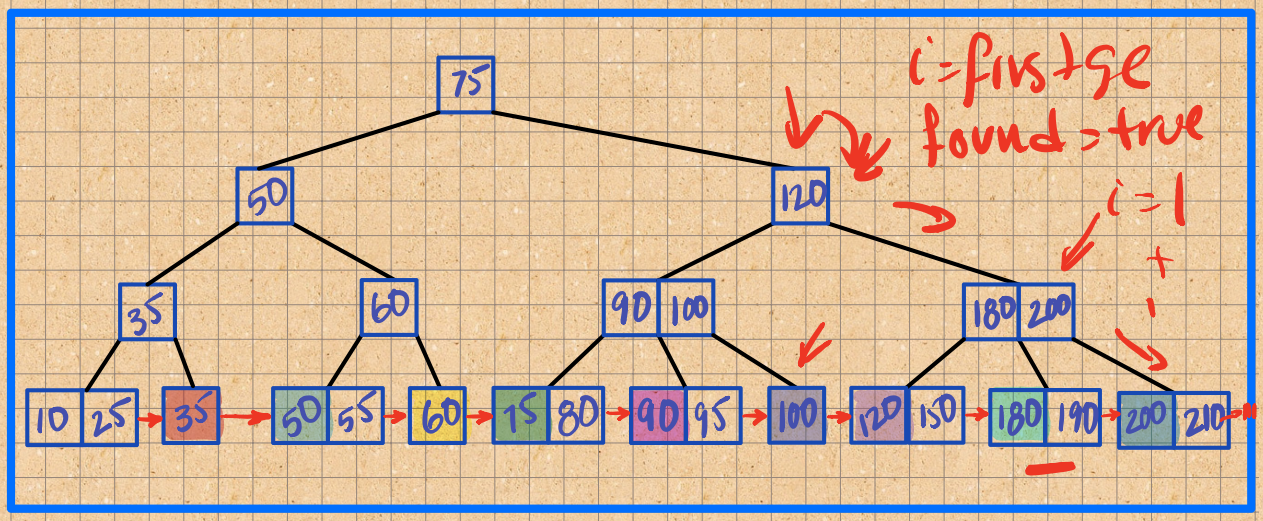


B Plus Tree

- All keys are present at the leaf level.
- A next pointer links each leaf node to the next and leaf nodes only
- Each entry in data[i] is equal to the first element of the leftmost leaf of subset [i+1]





- The inner nodes serve as bread crumbs to find the actual node in the leaves

General approach for b+trees to seek and process entries

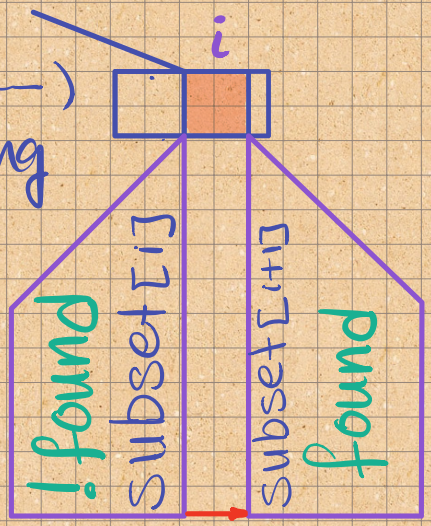
index = first_ge(...
found = ...

	found ✓	!found	
Leaf	found it	It's not there	inner nodes are bread crumbs:
!Leaf	subset[i+1] → recursive fix things	subset[i] → recursive fix things	

i = first_ge(—)

the key you are looking for is

in subset[i+1]
  if same key is in this inner node
else
  it could be in subset[i]

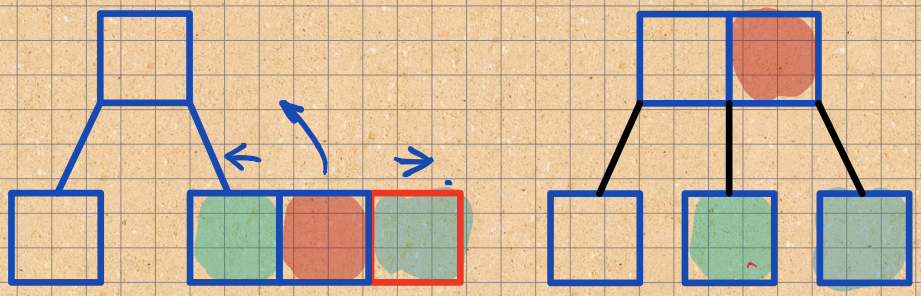


for loose_insert:

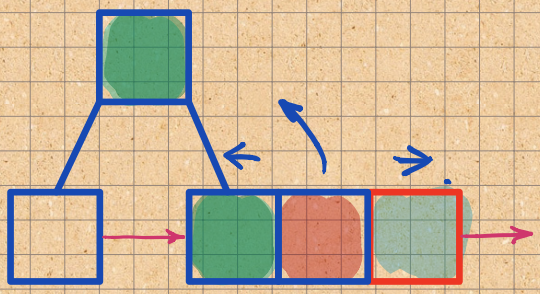
	found	!found	first_ge returns i found is true if i<dc & data[i] == entry
Leaf	it's a duplicate	insert key here @ pos i	
!Leaf	subset[i+1] → loose_insert fix-excess	subset[i] → loose_insert fix-excess	

fix_excess

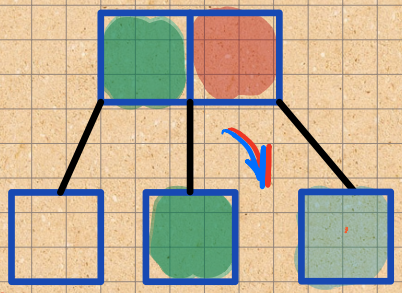
btree



b+tree

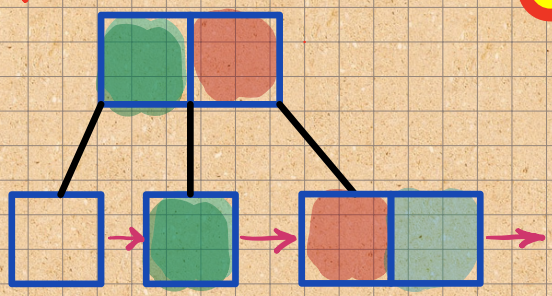


b+tree inner nodes behave exaclty as btree nodes do



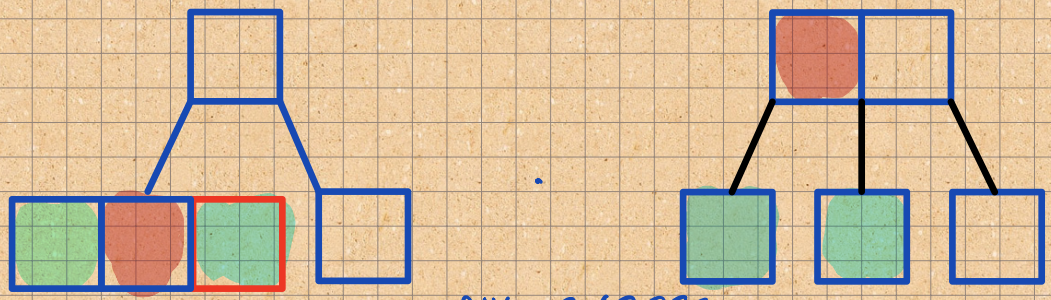
- Split and move half to i+1
  Move middle up to pos i

If a leaf:



- If leaf:
  insert middle value to pos 0 of i+1
  - connect the links

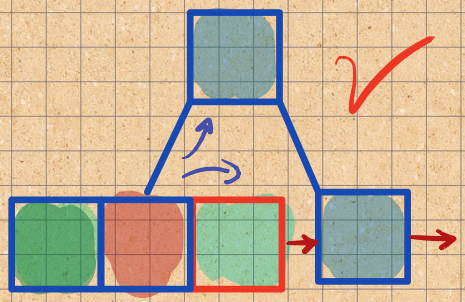
btree



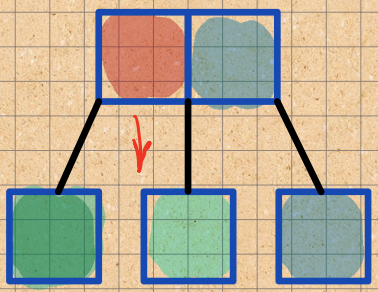
fix excess

b+tree

b+tree inner nodes behave exactly as btree nodes do

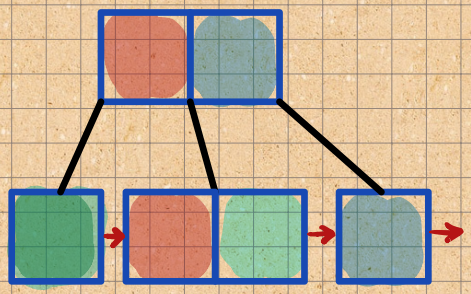


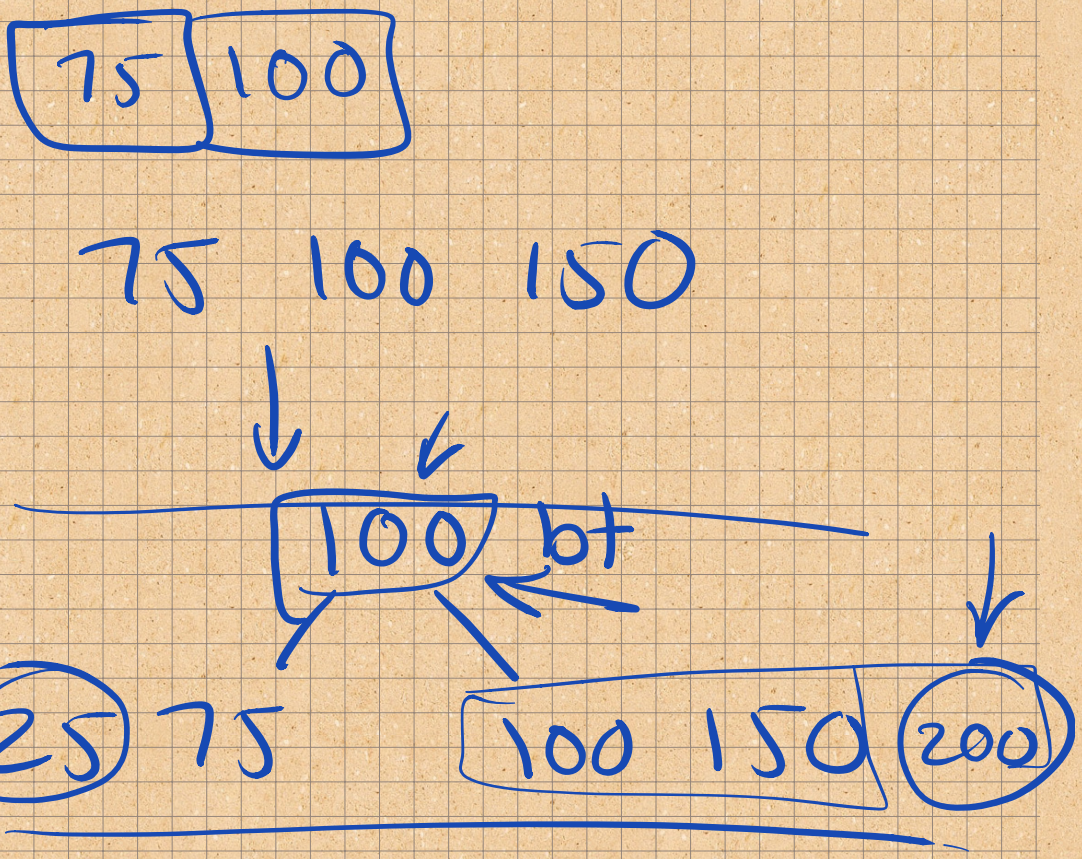
split & move middle value to position i in parent



if a leaf:
insert middle value at position 0 in i+1

fix links:
a standard insert_after



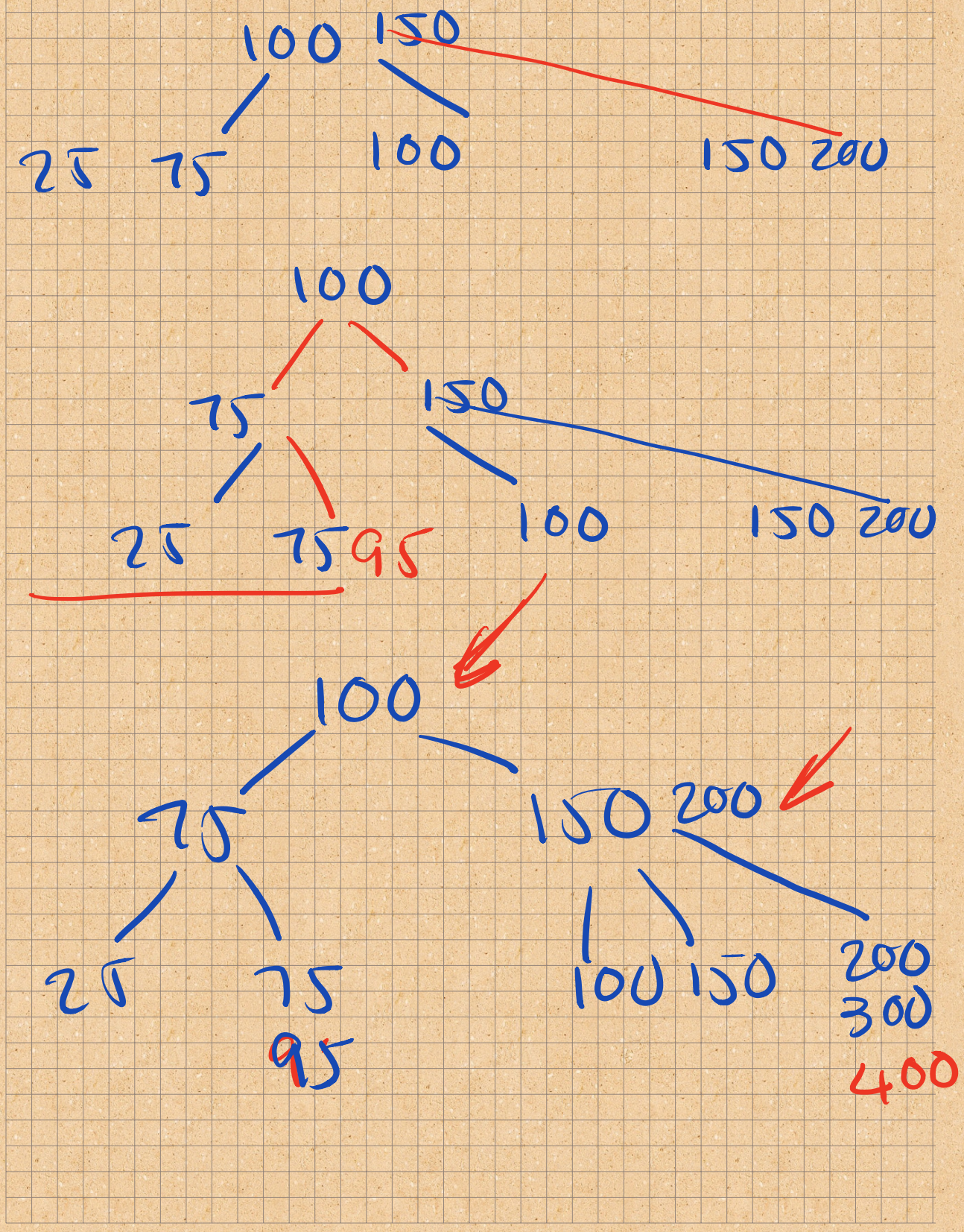


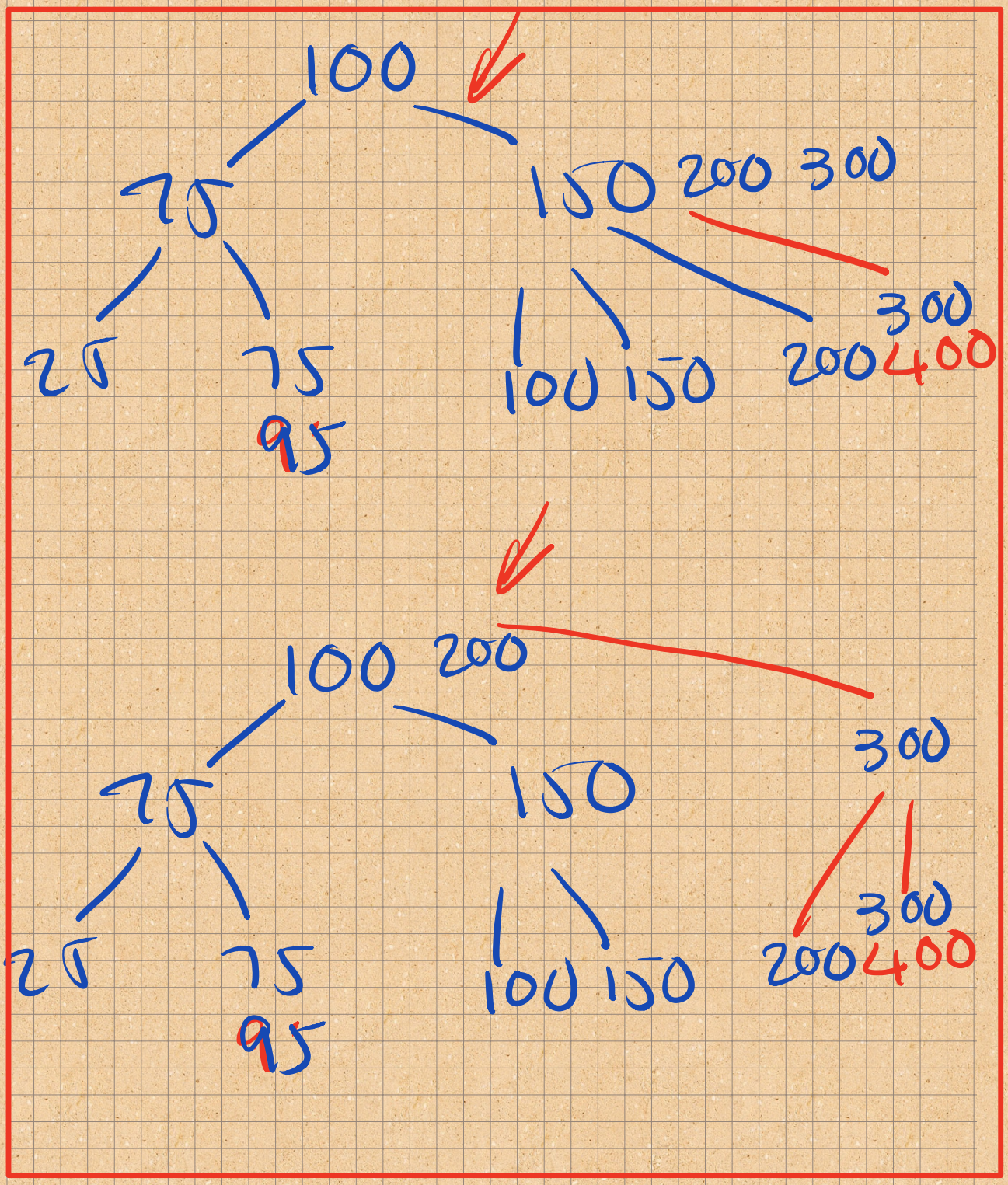
bt.insert(200)

i = fge( )

subset[1].insert

f.e.


100 150
25 75 100 150 200
100
75 150
25 75 95 100 150 200
100
75 150 200
25 75 95 100 150 200
300
400


100
75
150
200
300
25
75
95
100
150
200
300
400
100
200
75
150
300
25
75
95
100
150
200
300
400

loose_remove

may cause this node with less items than minimum

once loose_remove returns, the caller may need to call fix_shortage

- data to be removed resides in the leaf nodes
  - once the data is removed any "sister data" elements (breadcrums) must be removed and be replaced with the smallest data item in subset ith subtree
  - poorman's version:
    leaving the breadcrum data items on the tree will, in time, clutter the b+ tree, but will not interfere with the proper operations of the b+ tree - I hope.

fix_shortage:

fix-shortage is called on the child with less than minimum number of data items, by the parent node.

4 Scenarios:

1. child i+1 has more than min
   borrow from right:
   transfer / rotate left

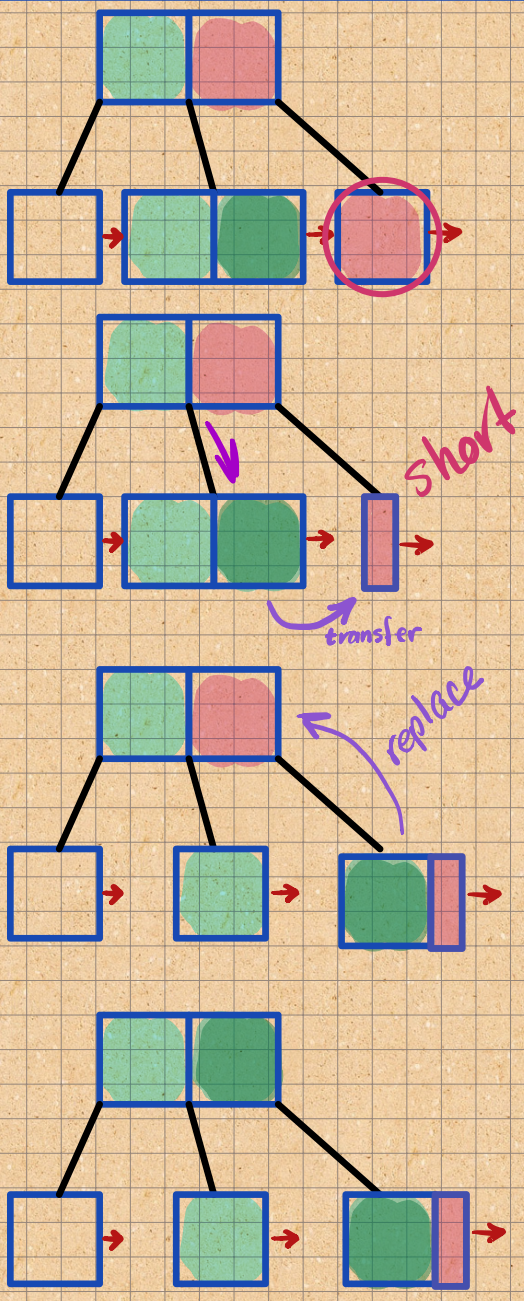
2. child i-1 has more than min
   borrow from left
   transfer / rotate right

3. there is a left sibling
   merge with previous:
   merge(i-1)

4. there has to be a right sibling
   merge with next:
   merge(i)

1. transfer right (i-1)

tell the subset to short's left (i-1), to transfer right to short

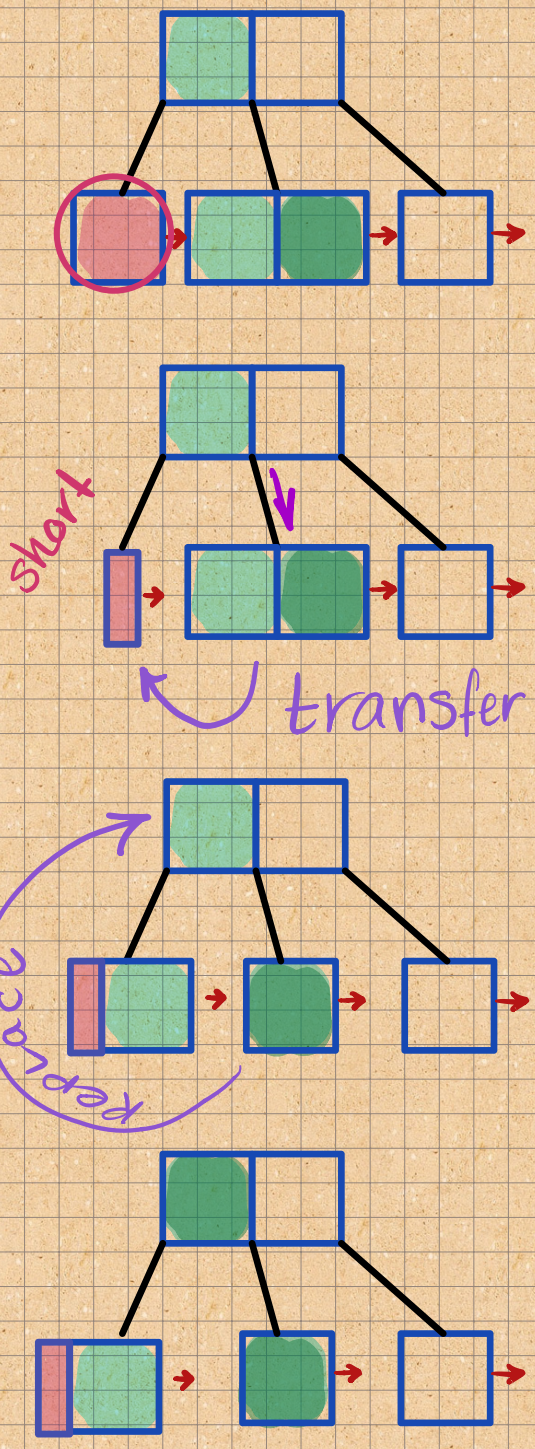


Leaves only

- transfer the last data item from [i-1]→data to [i]→data
- replace data[i] with this data item.

2. transfer Left (i+1)

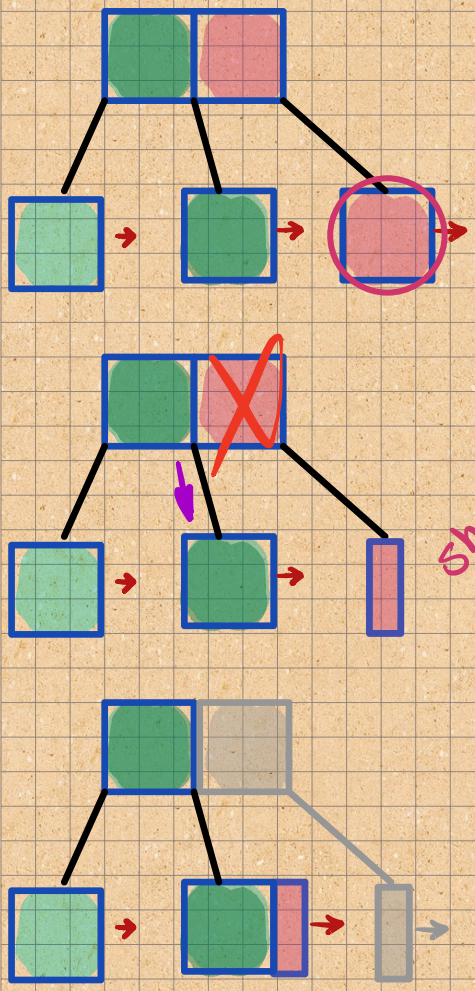
tell the subset to the left of short (i+1) to transfer left - to short



Leaves only

- Transfer
  [i+1] → data[Ø]
  to the end of
  [i] → data
- replace data[i]
  with
  [i+1] → data[Ø]

3. merge_with_next(i-1)



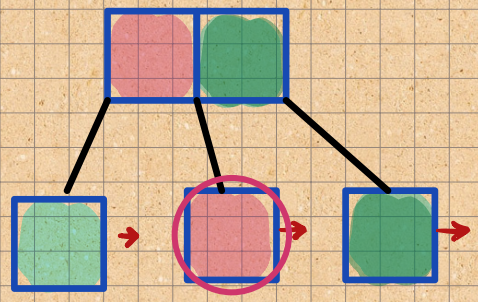
i>0: tell the subset to the left of short (i-1), to merge with his next subset - short

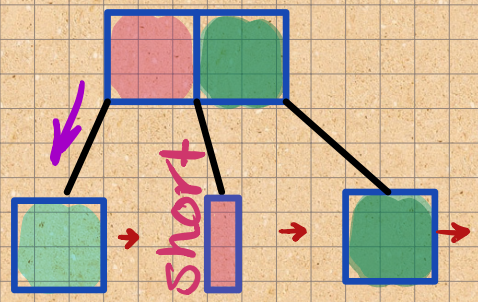
for leaves:

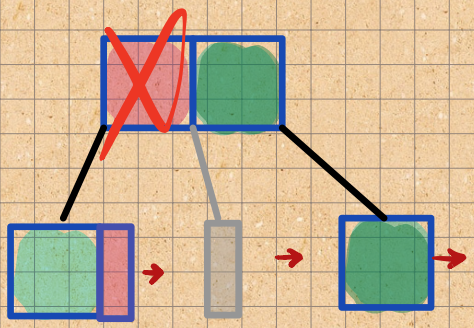
delete data[i], but do not attach to subset[i] → data

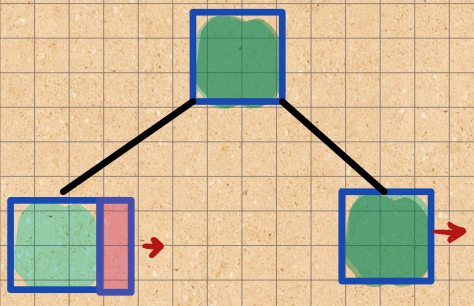
then merge data[i+1] with data[i]

4. merge_with_next(i)









i < data_count-1 :
merge the short subset with his next sibling

delete data[i], but
do not attach to
subset[i]->data

merge
subset[i+1]->data
with subset[i]->data

copy_tree

- A solid copy_tree is necessary for a reliable Big Three
- Challenge: How to connect the next pointers on the leaves
- copy_tree function takes two arguments:
  - a btree node : Other
  - a btree pointer: (by reference) "last_leaf"
- copy_list algorithm

copy the other's data over
if the other guy is not a leaf
  for every child of the other
    Copy other's child as my own

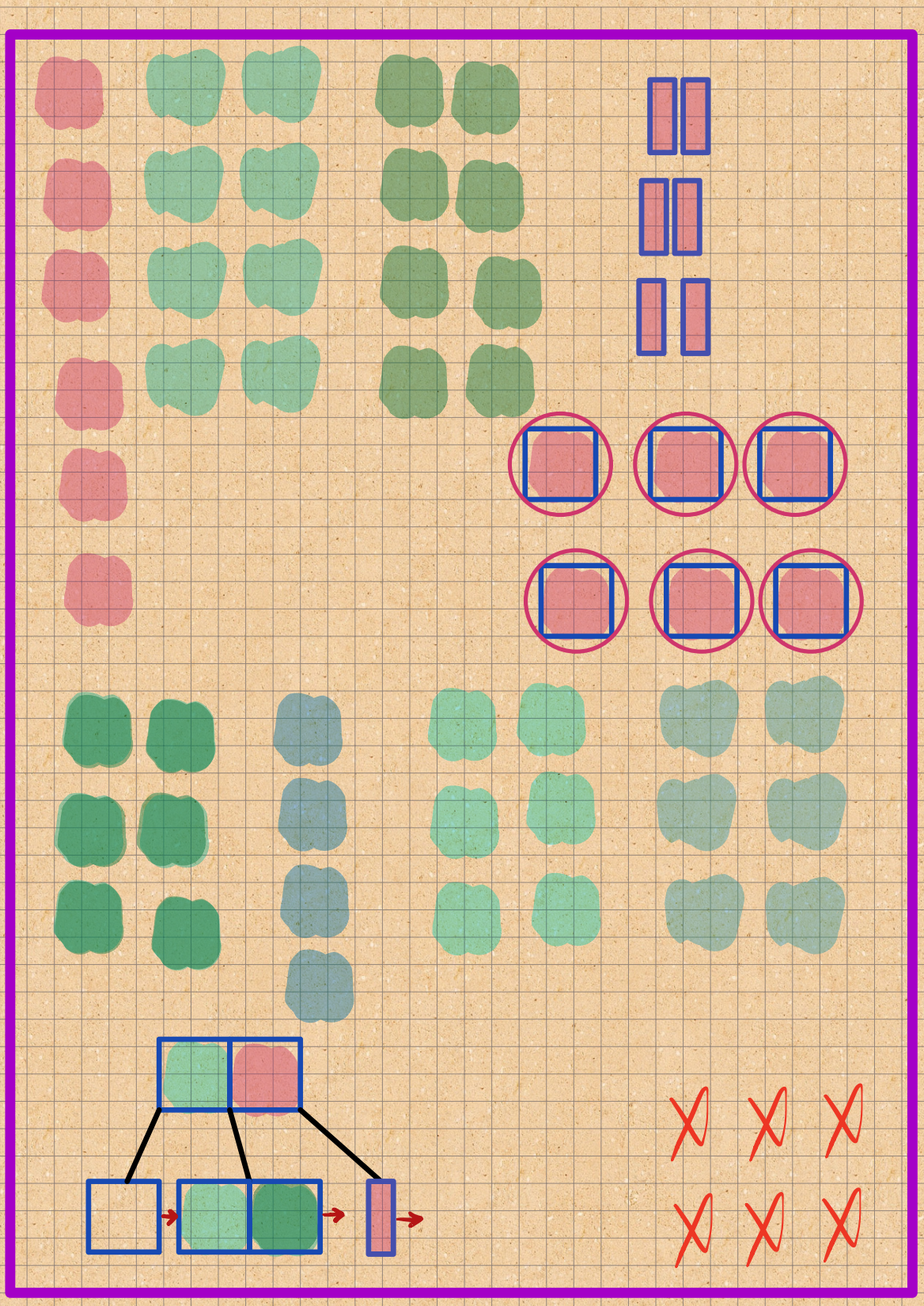
else, if other is a leaf:
  point last_leaf's next to this node if last_leaf is not null
  set last_leaf to this node.

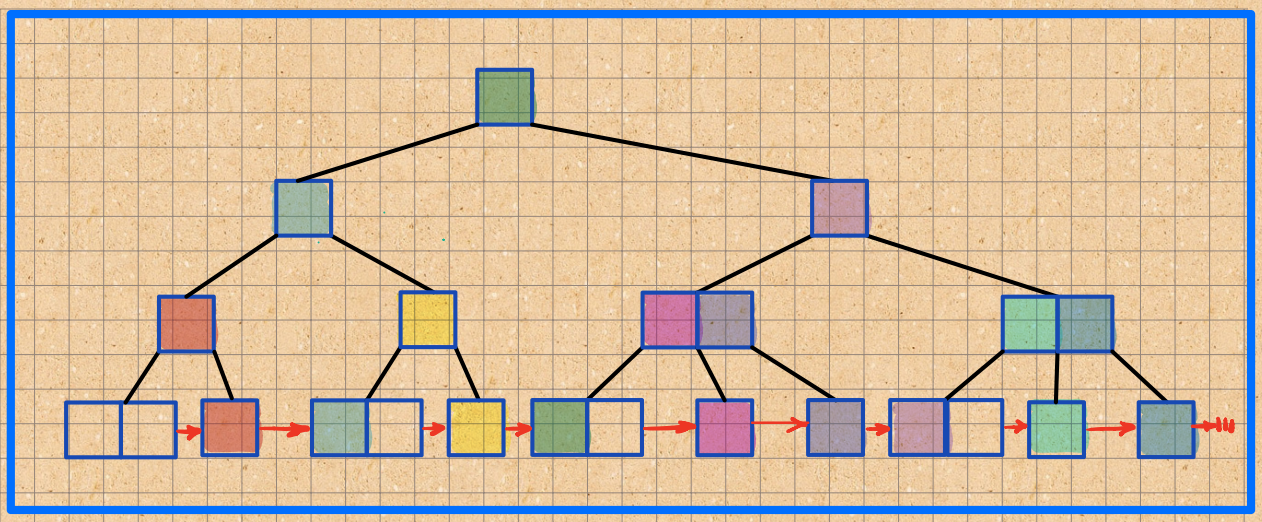
Iterators

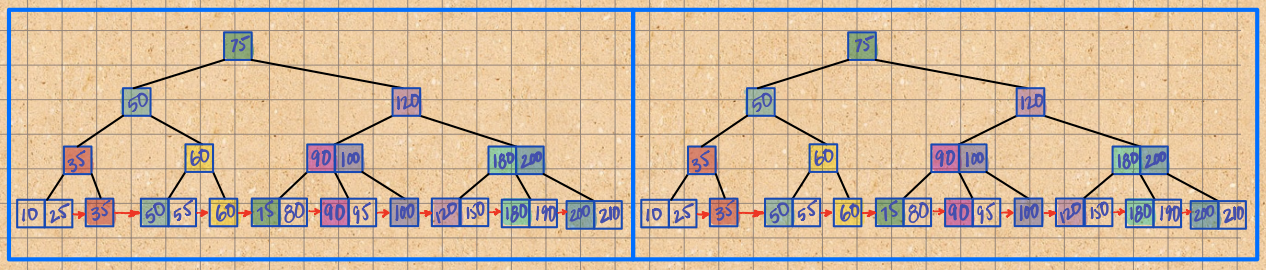
A pointer for the nodes
and
an int for the data index in data array

++ : increment the int until it reaches dc-1 then set it to zero & advance the pointer

* : returns T at index of the node being pointed to by pointer






75
50
120
35
60
90
100
180
200
10
25
35
50
55
60
75
80
90
95
100
120
150
180
190
200
210