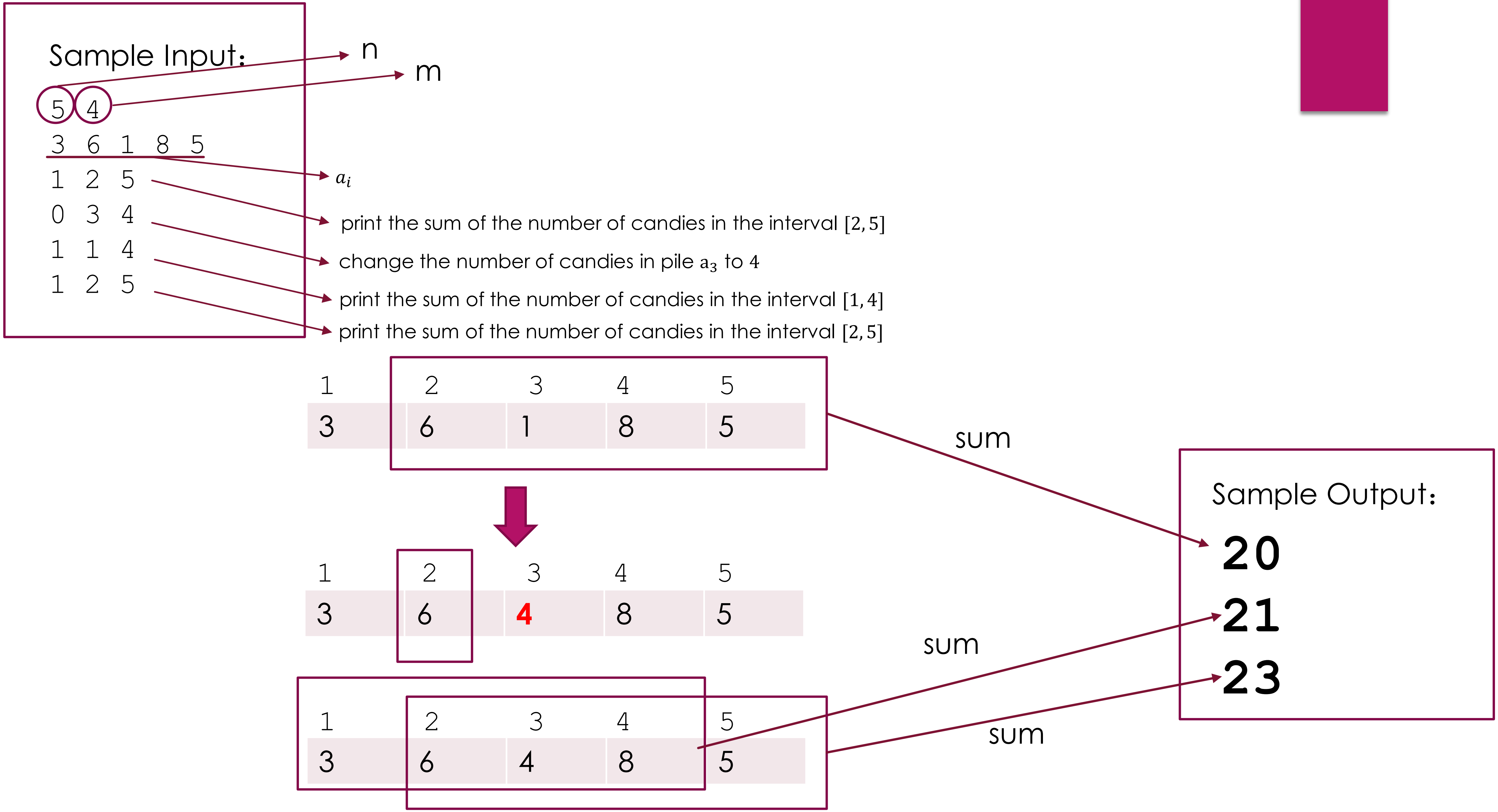
Lab9 Solution

YAO ZHAO

Lab9.A Candy

- Recently, Andrea received n piles of candy. The $i-th$ pile contains a_i candies. She would perform m operations on these piles as follows:
  - $0\ x\ v$: Change the number of candies in pile a_x to v
  - $1\ l\ r$: print the sum of the number of candies in the given interval $[l,r]$



Sample Input:

```
5 4
3 6 1 8 5
1 2 5
0 3 4
1 1 4
1 2 5
```

print the sum of the number of candies in the interval $[2,5]$

change the number of candies in pile a_3 to 4

print the sum of the number of candies in the interval $[1,4]$

print the sum of the number of candies in the interval $[2,5]$

1	2	3	4	5
3	6	1	8	5

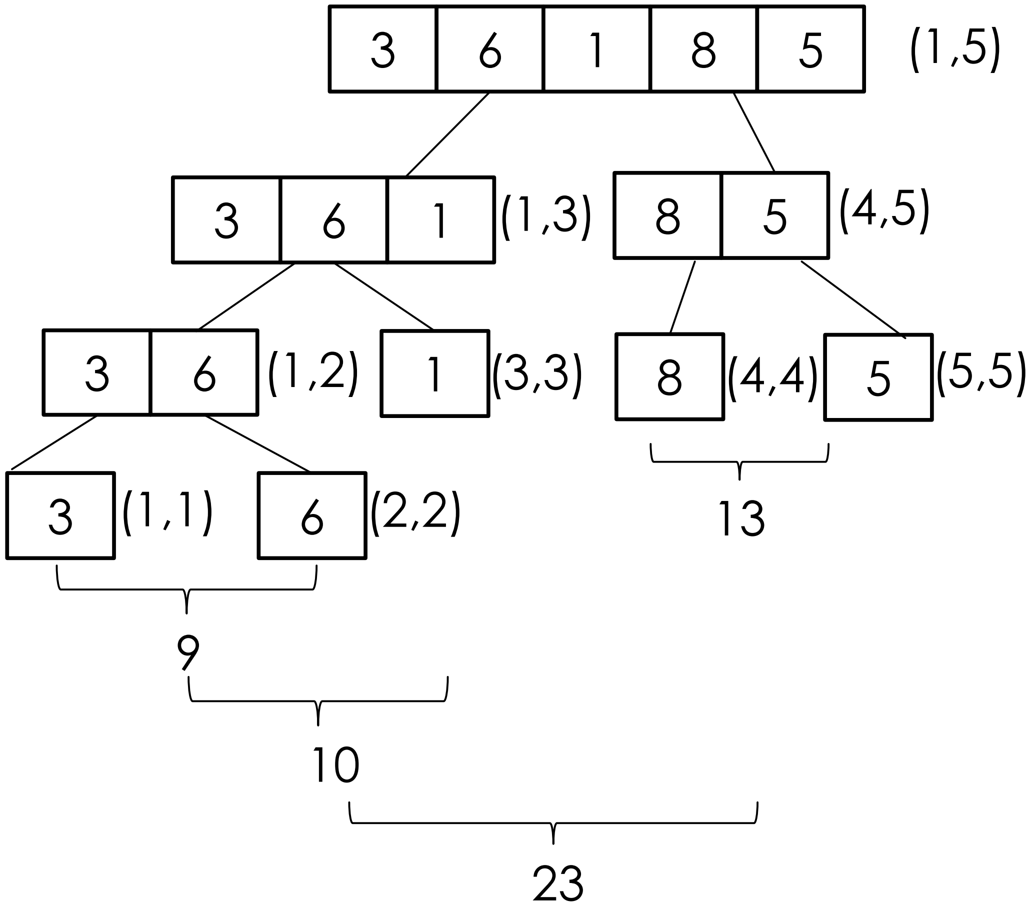
1	2	3	4	5
3	6	4	8	5

1	2	3	4	5
3	6	4	8	5

Sample Output:

```
20
21
23
```

parent node index : i
children nodes index:
left: 2*i
right: 2*i+1



Build tree

1	2	3	4	5	6	7	8	9
l:1 r:5 sum:23	l:1 r:3 sum:10	l:4 r:5 sum:13	l:1 r:2 sum:9	l:3 r:3 sum:1	l:4 r:4 sum:8	l:5 r:5 sum:5	l:1 r:1 sum:3	l:2 r:2 sum:6

1 2 5 calculate $\sum_{i=2}^{5} a_i$

1	2	3	4	5	6	7	8	9
l:1 r:5 sum:23	l:1 r:3 sum:10	l:4 r:5 sum:13	l:1 r:2 sum:9	l:3 r:3 sum:1	l:4 r:4 sum:8	l:5 r:5 sum:5	l:1 r:1 sum:3	l:2 r:2 sum:6

Step1: search from root **index = 1**
[2,5]⊃ [1,5] ? No
Left node(**index:2**): [1,3] ∩ [2,5] ≠ ∅ search left node
Right node(**index:3**): [4,5] ∩ [2,5] ≠ ∅ search right node

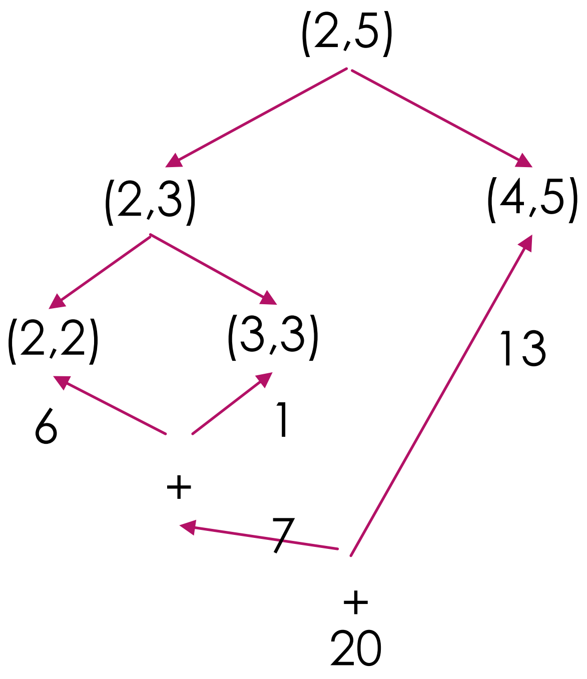
Step 2: **index = 2**
[2,5] ⊃ [1,3] ?No
Left node(**index:4**): [1,2] ∩ [2,5] ≠ ∅ search left node
Right node(**index:5**): [3,3] ∩ [2,5] ≠ ∅ search right node

Step 3: **index = 4**
[2,5] ⊃ [1,2]? No
Left node(**index:8**): [1,1] ∩ [2,5] = ∅ stop
Right node(**index:9**): [2,2] ∩ [2,5] ≠ ∅ search right node

Step 4: **index = 9**
[2,5] ⊃ [2,2]? Yes
return 6

Step 5: **index = 5**
[2,5] ⊃ [3,3]? Yes
return 1

Step 6: **index = 3**
[2,5] ⊃ [4,5]? Yes
return 13



0 3 4 change a_3 to 4

1	2	3	4	5	6	7	8	9
l:1 r:5 sum:28	l:1 r:3 sum:10	l:4 r:5 sum:13	l:1 r:2 sum:9	l:3 r:3 sum:4	l:4 r:4 sum:8	l:5 r:5 sum:5	l:1 r:1 sum:3	l:2 r:2 sum:6

Step1: target index = 3
mid = (1+5)/2 = 3
Left node: target index ≤ mid ? **Yes: search left node**
No: search right node

Step 2:target index = 3
mid = (1+3)/2 = 2
Left node: target index ≤ mid ? Yes: search left node
No: search right node

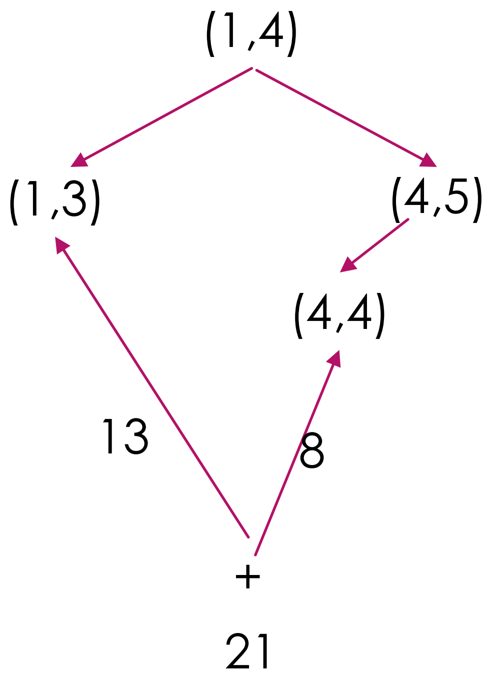
Step 3:target index =3
l = r = target index
Leaf node: set sum = 4
Then update the parent nodes of the leaf node

0 3 4 change a_3 to 4

	1	2	3	4	5	6	7	8	9
Result:	l:1 r:5 sum:**26**	l:1 r:3 sum:**13**	l:4 r:5 sum:13	l:1 r:2 sum:9	l:3 r:3 sum:**4**	l:4 r:4 sum:8	l:5 r:5 sum:5	l:1 r:1 sum:3	l:2 r:2 sum:6

1 1 4 calculate $\sum_{i=1}^{4} a_i$

1	2	3	4	5	6	7	8	9
l:1 r:5 sum:**26**	l:1 r:3 sum:**13**	l:4 r:5 sum:13	l:1 r:2 sum:9	l:3 r:3 sum:**4**	l:4 r:4 sum:8	l:5 r:5 sum:5	l:1 r:1 sum:3	l:2 r:2 sum:6



Lab9.B Hotel

- The city where Andrea lives is a beautiful place that attracts many tourists. All the tourists will check into the biggest hotel in this city.
- The hotel has n rooms and receives m check in/checkout requests in the holiday. The rooms are numbered from 1 to n and at the beginning, all rooms are empty.
- The tourists always come in groups and when the $i-th$ group checks in, they need x contiguous rooms. If there are lots of rooms that meet the request, they will choose the room number to be the smallest possible and check in.
- The tourists also check out in groups and when the $i-th$ group checks out, the rooms numbered from l_i to r_i will be empty. Please note that some of those rooms may have been empty before the request.
- Please help the hotel deal with these requests. For each check in request, please tell the tourists the number of the leftmost room they can check in.
  - $1\ x$: The check in request.
  - $2\ l\ r$: The checkout request.

Sample Input:

```
5 5
1 3
1 2
2 3 5
2 1 1
1 3
```

n: the number of rooms

m: the number of requests

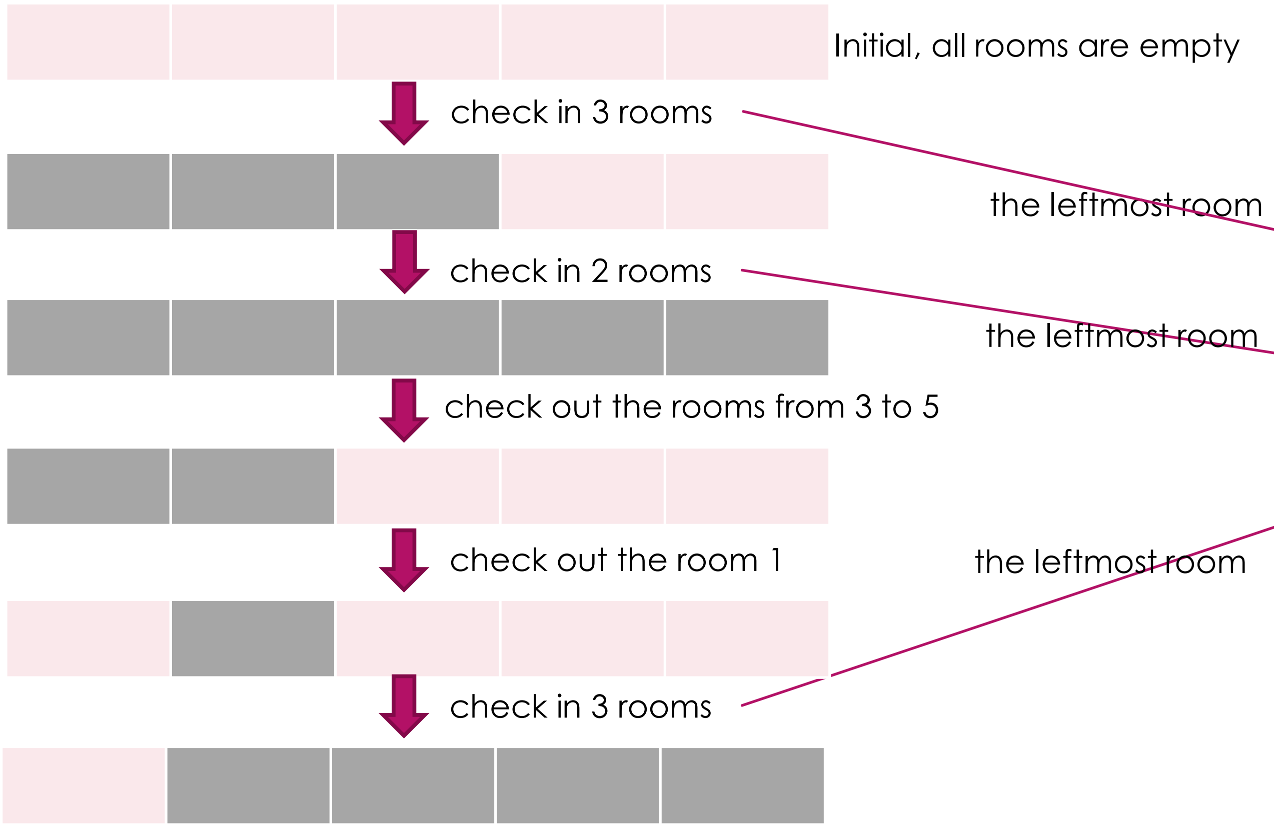
check in 3 rooms

check in 2 rooms

check out the rooms from 3 to 5

check out the room 1

check in 3 rooms



Initial, all rooms are empty

check in 3 rooms

the leftmost room

check in 2 rooms

the leftmost room

check out the rooms from 3 to 5

check out the room 1

the leftmost room

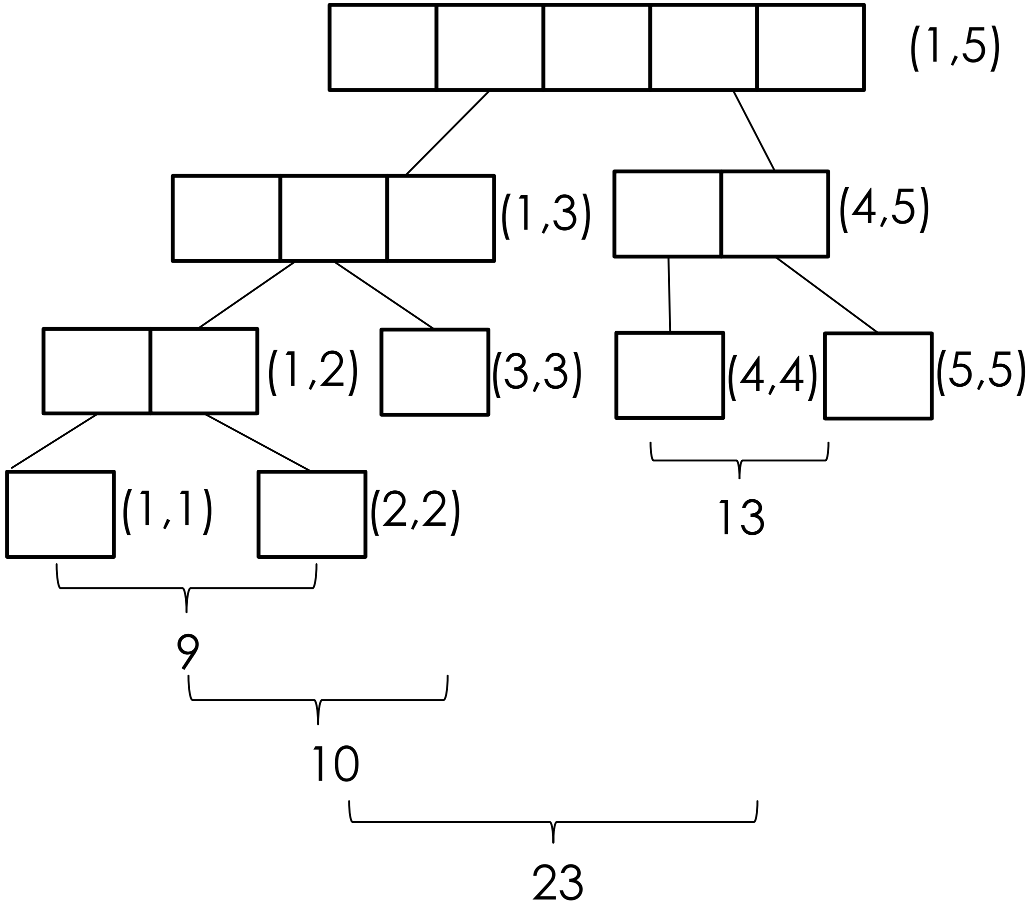
check in 3 rooms

Sample Output:

```
1
4
3
```

Question A and Question B are similar, the difference is:

1. question A is querying sum and question B is querying the left point of A free interval of the specified length.
2. question A only changes 1 element in an operation, and question B will change all elements in an interval.



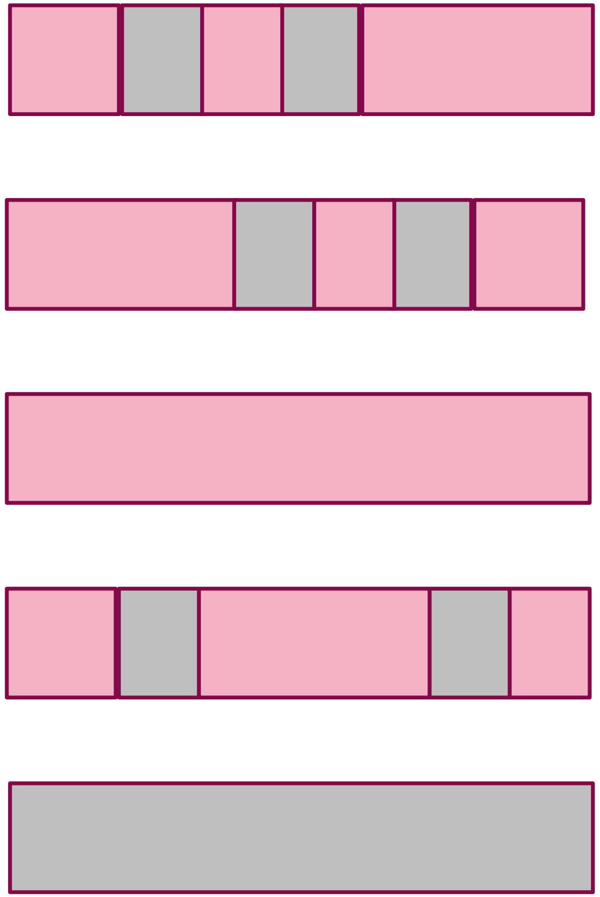
parent node index : i
children nodes index:
left: 2*i
right: 2*i+1

Build tree

1	2	3	4	5	6	7	8	9
l:1 r:5 l free len:? r free len:? max len:?	l:1 r:3 l free len:? r free len:? max len:?	l:4 r:5 l free len:? r free len:? max len:?	l:1 r:2 l free len:? r free len:? max len:?	l:3 r:3 l free len:1 r free len:1 max len:1	l:4 r:4 l free len:1 r free len:1 max len:1	l:5 r:5 l free len:1 r free len:1 max len:1	l:1 r:1 l free len:1 r free len:1 max len:1	l:2 r:2 l free len:1 r free len:1 max len:1

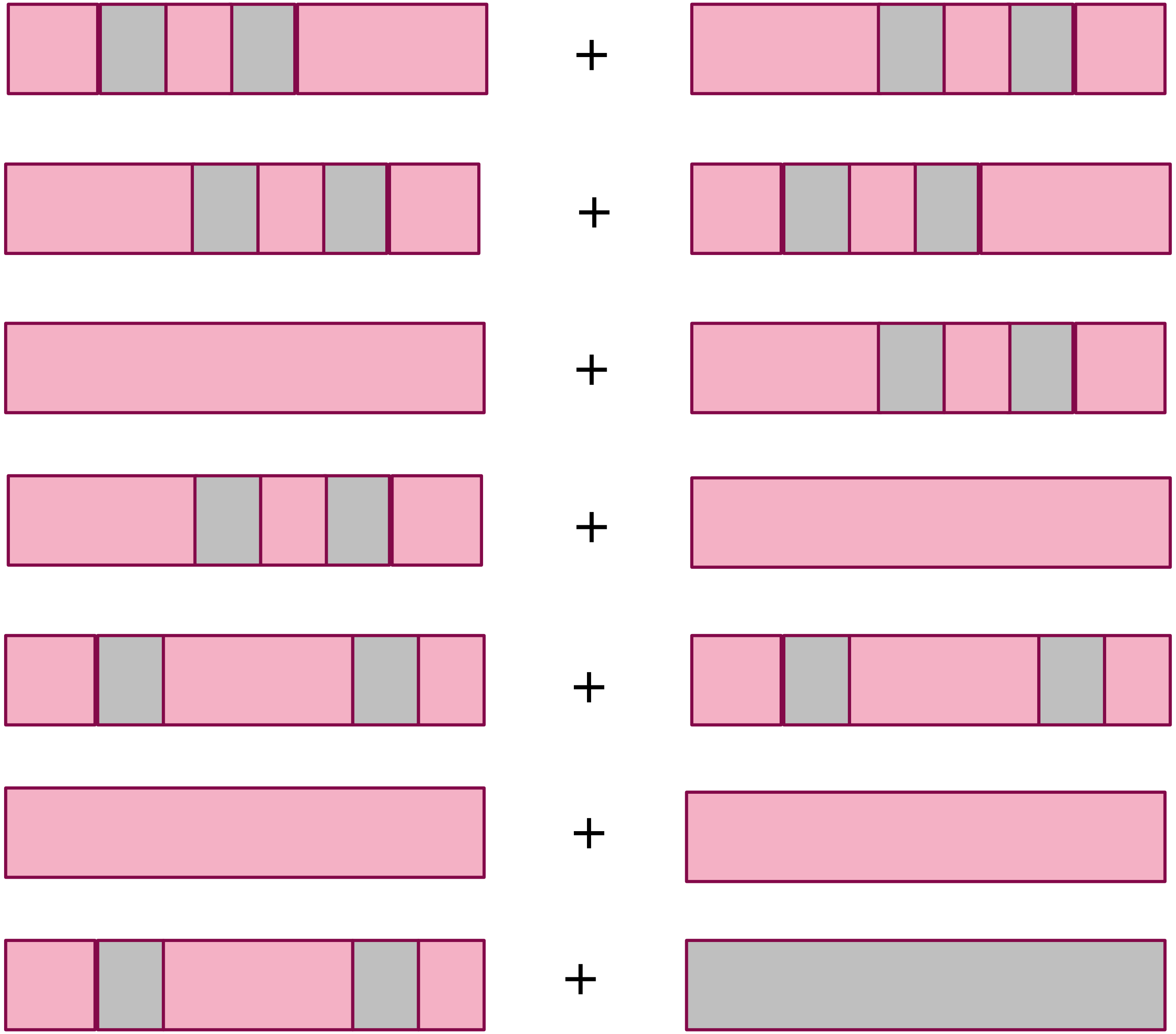
l:1 r:5 l free len:5 r free len:5 max len:5	l:1 r:3 l free len:3 r free len:3 max len:3	l:4 r:5 l free len:2 r free len:2 max len:2	l:1 r:2 l free len:2 r free len:2 max len:2

How to calcu the l free len , r free len and max free len from children?

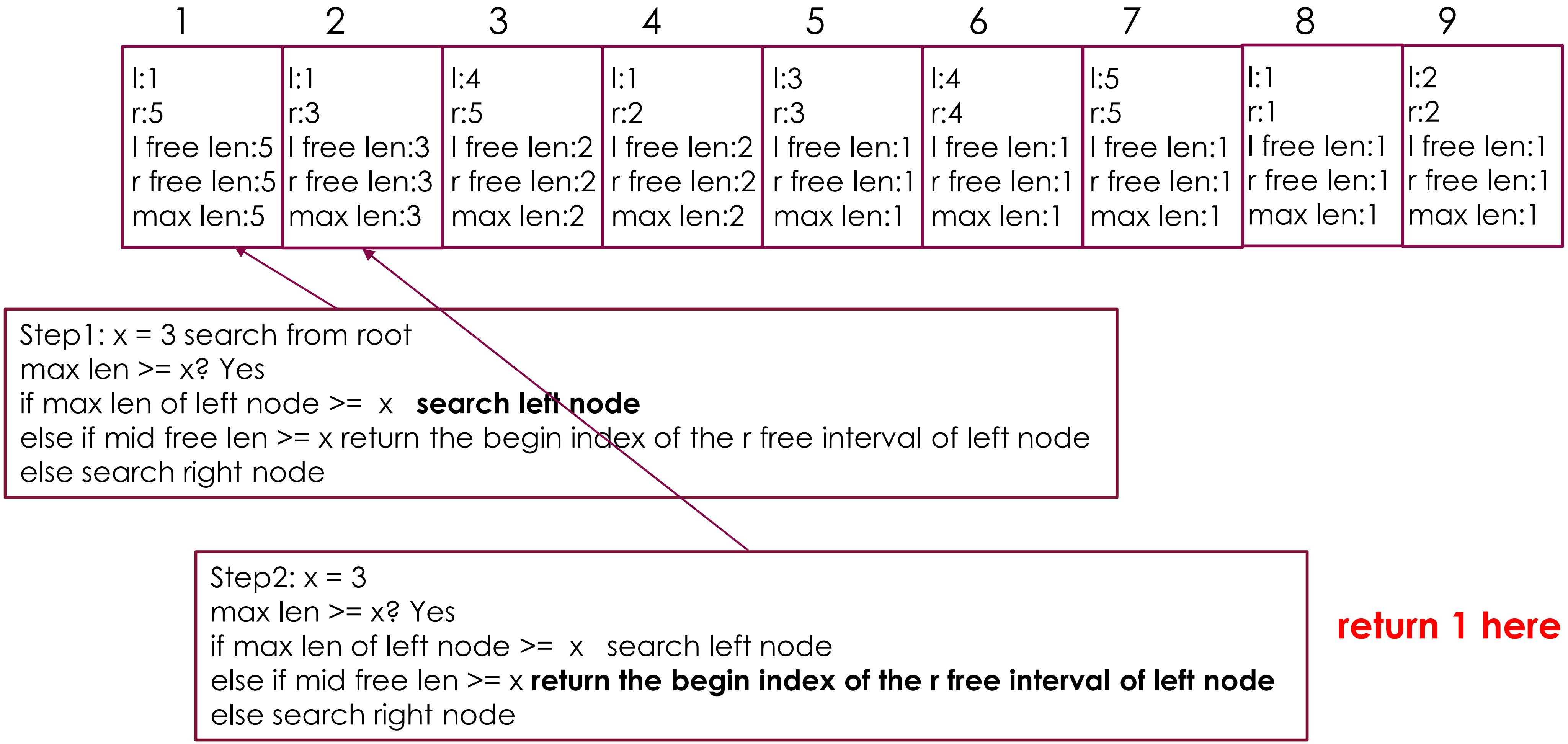


left child

right child



1 3 check in 3 rooms querying the left point of a free interval of the specified length



1	2	3	4	5	6	7	8	9
l:1	l:1	l:4	l:1	l:3	l:4	l:5	l:1	l:2
r:5	r:3	r:5	r:2	r:3	r:4	r:5	r:1	r:2
l free len:5	l free len:3	l free len:2	l free len:2	l free len:1	l free len:1	l free len:1	l free len:1	l free len:1
r free len:5	r free len:3	r free len:2	r free len:2	r free len:1	r free len:1	r free len:1	r free len:1	r free len:1
max len:5	max len:3	max len:2	max len:2	max len:1	max len:1	max len:1	max len:1	max len:1

Step1: x = 3 search from root
max len >= x? Yes
if max len of left node >= x **search left node**
else if mid free len >= x return the begin index of the r free interval of left node
else search right node

Step2: x = 3
max len >= x? Yes
if max len of left node >= x search left node
else if mid free len >= x **return the begin index of the r free interval of left node**
else search right node

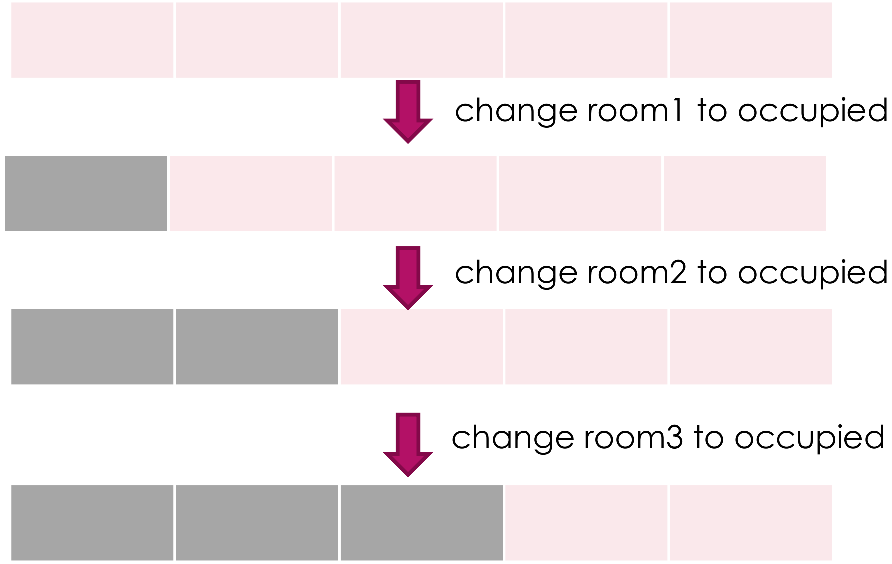
return 1 here

output 1

then change the room 1~ 3 to “occupied”

1	2	3	4	5	6	7	8	9
l:1	l:1	l:4	l:1	l:3	l:4	l:5	l:1	l:2
r:5	r:3	r:5	r:2	r:3	r:4	r:5	r:1	r:2
l free len:5	l free len:3	l free len:2	l free len:2	l free len:1	l free len:1	l free len:1	l free len:1	l free len:1
r free len:5	r free len:3	r free len:2	r free len:2	r free len:1	r free len:1	r free len:1	r free len:1	r free len:1
max len:5	max len:3	max len:2	max len:2	max len:1	max len:1	max len:1	max len:1	max len:1

You can do the same things on page 6.
One time change one value, repeat 3 times



change room1 to occupied

change room2 to occupied

change room3 to occupied

X But this is inefficient

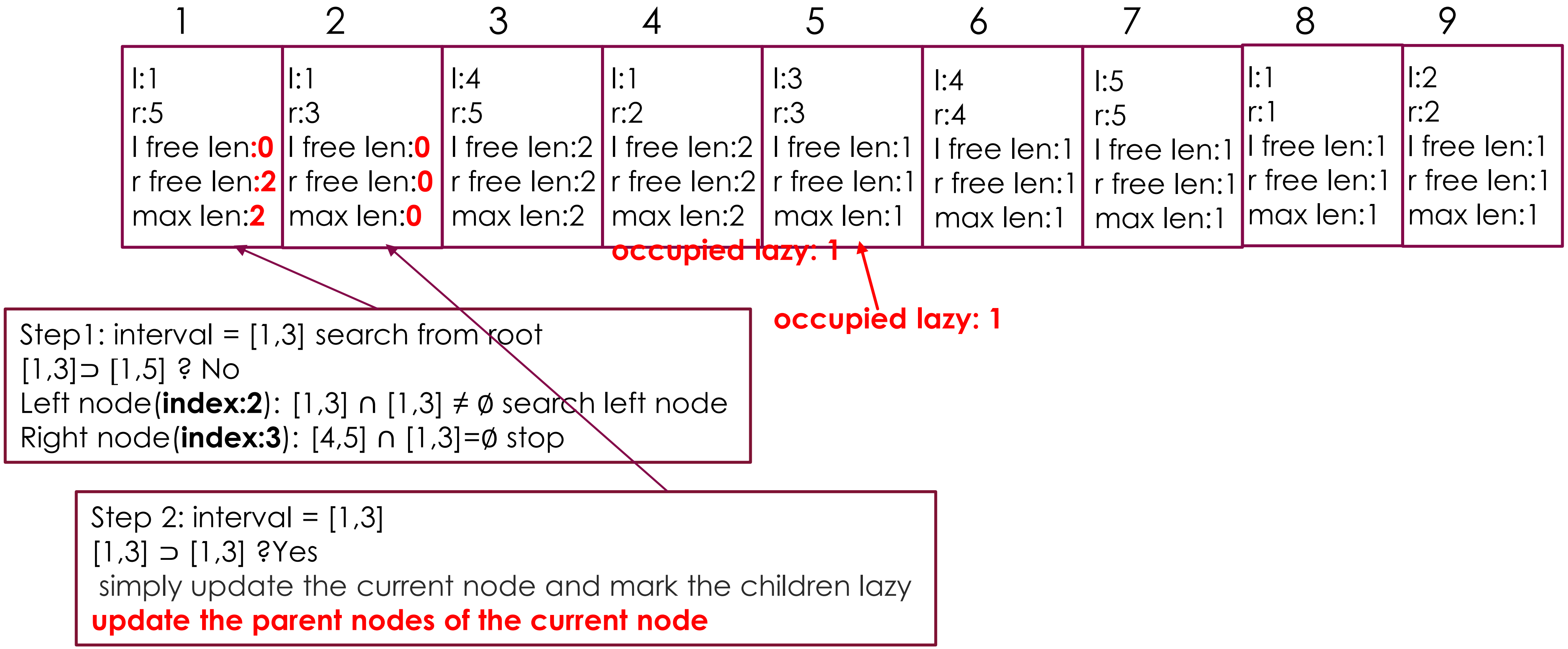
LAZY PROPAGATION

Lazy propagation is a range update and query optimized implementation of a segment tree that performs both operation O(logN) time.
In short, as the name suggests, the algorithm works on laziness for what is not important at the time. Don't update a node until needed, which will avoid the repeated sharing.

CASES WHILE UPDATING:
1.Segment lies outside the query range: in this case, we can just simply return back and terminate the call.
2.Segment lies fully inside the query range: in this case, we simply update the current node and mark the children lazy.
3.If they intersect partially, then we all update for both the child and change the values in them.

https://www.topcoder.com/thrive/articles/range-operations-lazy-propagation#:~:text=Lazy%20propagation%20is%20a%20range,will%20avoid%20the%20repeated%20sharing.

change the room 1~ 3 to “occupied”



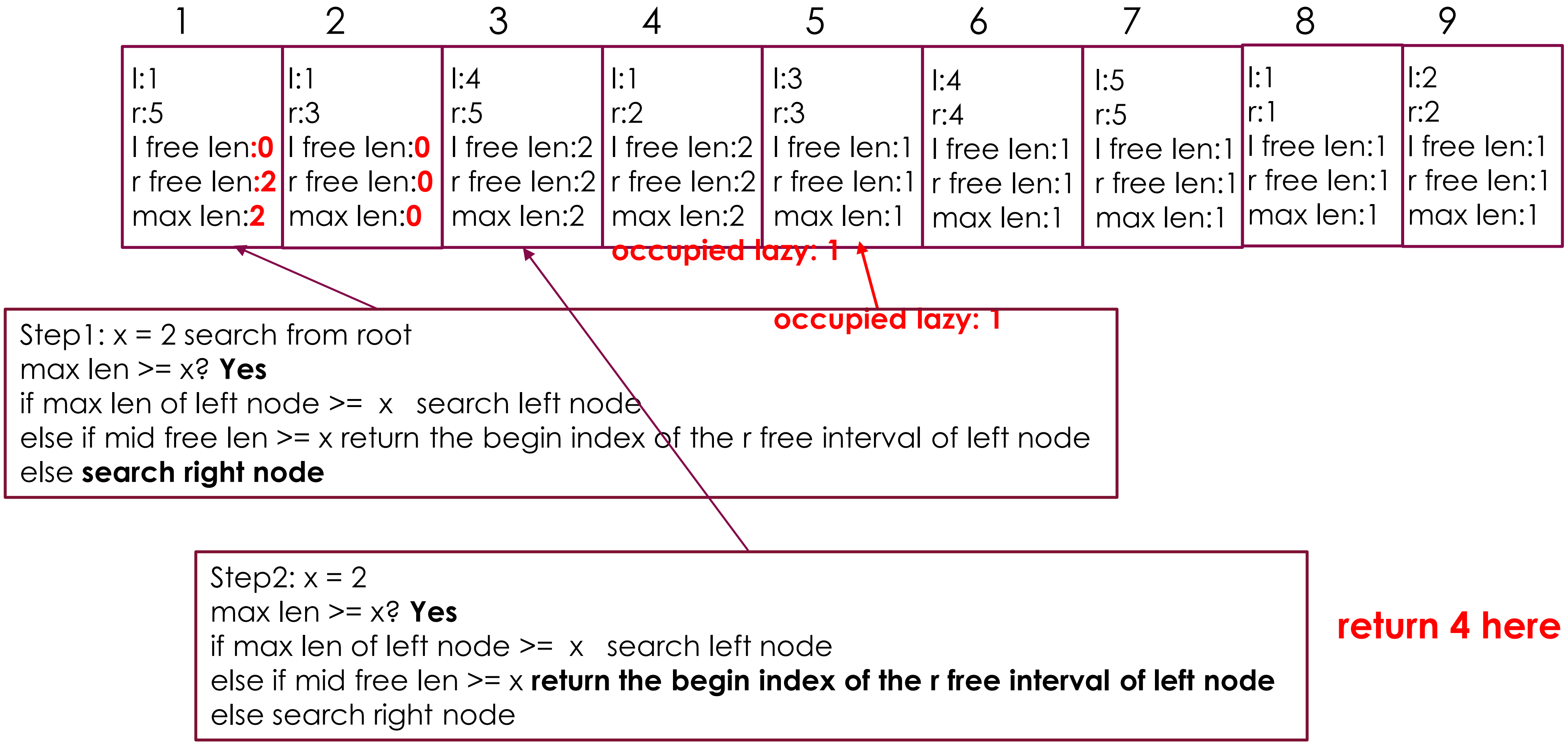
1	2	3	4	5	6	7	8	9
l:1	l:1	l:4	l:1	l:3	l:4	l:5	l:1	l:2
r:5	r:3	r:5	r:2	r:3	r:4	r:5	r:1	r:2
l free len:**0**	l free len:**0**	l free len:2	l free len:2	l free len:1	l free len:1	l free len:1	l free len:1	l free len:1
r free len:**2**	r free len:**0**	r free len:2	r free len:2	r free len:1	r free len:1	r free len:1	r free len:1	r free len:1
max len:**2**	max len:**0**	max len:2	max len:2	max len:1	max len:1	max len:1	max len:1	max len:1
			occupied lazy: 1					

occupied lazy: 1

Step1: interval = [1,3] search from root
[1,3]⊃ [1,5] ? No
Left node(**index:2**): [1,3] ∩ [1,3] ≠ Ø search left node
Right node(**index:3**): [4,5] ∩ [1,3]=Ø stop

Step 2: interval = [1,3]
[1,3] ⊃ [1,3] ?Yes
simply update the current node and mark the children lazy
update the parent nodes of the current node

1 2 check in 2 rooms querying the left point of a free interval of the specified length



1	2	3	4	5	6	7	8	9
l:1 r:5 l free len:0 r free len:2 max len:2	l:1 r:3 l free len:0 r free len:0 max len:0	l:4 r:5 l free len:2 r free len:2 max len:2	l:1 r:2 l free len:2 r free len:2 max len:2	l:3 r:3 l free len:1 r free len:1 max len:1	l:4 r:4 l free len:1 r free len:1 max len:1	l:5 r:5 l free len:1 r free len:1 max len:1	l:1 r:1 l free len:1 r free len:1 max len:1	l:2 r:2 l free len:1 r free len:1 max len:1

occupied lazy: 1

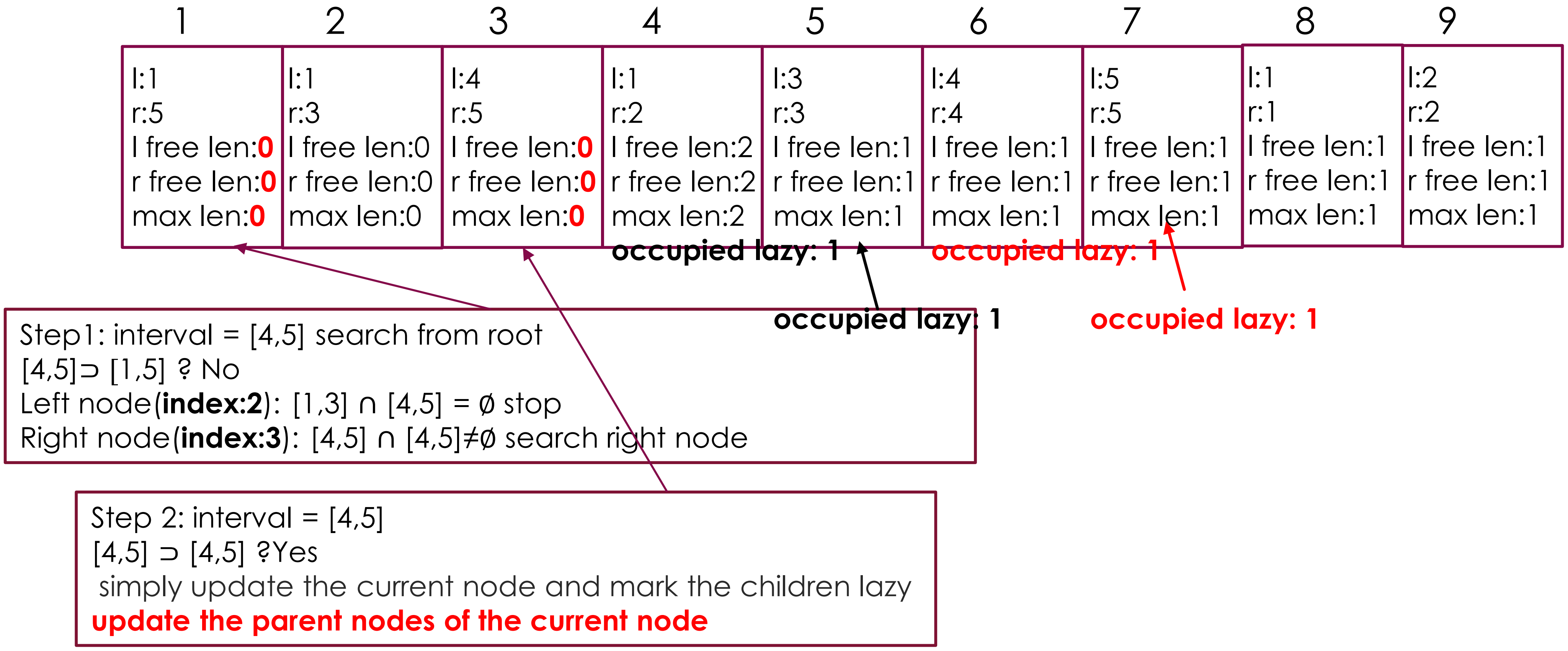
occupied lazy: 1

Step1: x = 2 search from root
max len >= x? **Yes**
if max len of left node >= x search left node
else if mid free len >= x return the begin index of the r free interval of left node
else **search right node**

Step2: x = 2
max len >= x? **Yes**
if max len of left node >= x search left node
else if mid free len >= x **return the begin index of the r free interval of left node**
else search right node

return 4 here

change the room 4~ 5 to “occupied”



1	2	3	4	5	6	7	8	9
l:1 r:5 l free len:**0** r free len:**0** max len:**0**	l:1 r:3 l free len:0 r free len:0 max len:0	l:4 r:5 l free len:**0** r free len:**0** max len:**0**	l:1 r:2 l free len:2 r free len:2 max len:2 **occupied lazy: 1**	l:3 r:3 l free len:1 r free len:1 max len:1	l:4 r:4 l free len:1 r free len:1 max len:1 **occupied lazy: 1**	l:5 r:5 l free len:1 r free len:1 max len:1	l:1 r:1 l free len:1 r free len:1 max len:1	l:2 r:2 l free len:1 r free len:1 max len:1

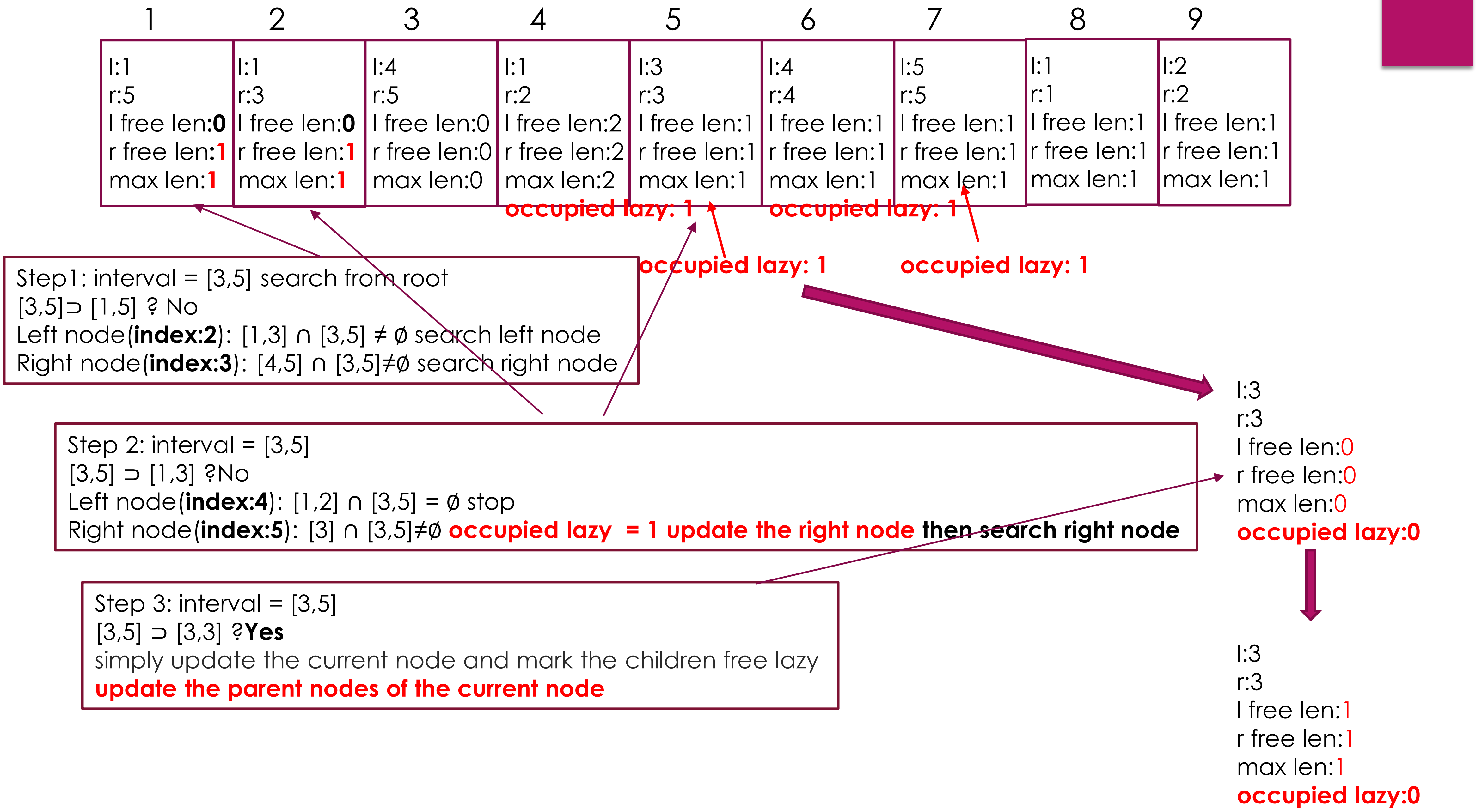
occupied lazy: 1

occupied lazy: 1

Step1: interval = [4,5] search from root
[4,5]⊃ [1,5] ? No
Left node(**index:2**): [1,3] ∩ [4,5] = ∅ stop
Right node(**index:3**): [4,5] ∩ [4,5]≠∅ search right node

Step 2: interval = [4,5]
[4,5] ⊃ [4,5] ?Yes
simply update the current node and mark the children lazy
update the parent nodes of the current node

2 3 5 check out room 3,4,5



1	2	3	4	5	6	7	8	9
l:1	l:1	l:4	l:1	l:3	l:4	l:5	l:1	l:2
r:5	r:3	r:5	r:2	r:3	r:4	r:5	r:1	r:2
l free len:**0**	l free len:**0**	l free len:0	l free len:2	l free len:1	l free len:1	l free len:1	l free len:1	l free len:1
r free len:**1**	r free len:**1**	r free len:0	r free len:2	r free len:1	r free len:1	r free len:1	r free len:1	r free len:1
max len:**1**	max len:**1**	max len:0	max len:2	max len:1	max len:1	max len:1	max len:1	max len:1
			occupied lazy: 1		**occupied lazy: 1**			

occupied lazy: 1

occupied lazy: 1

Step1: interval = [3,5] search from root
[3,5]⊃ [1,5] ? No
Left node(**index:2**): [1,3] ∩ [3,5] ≠ ∅ search left node
Right node(**index:3**): [4,5] ∩ [3,5]≠∅ search right node

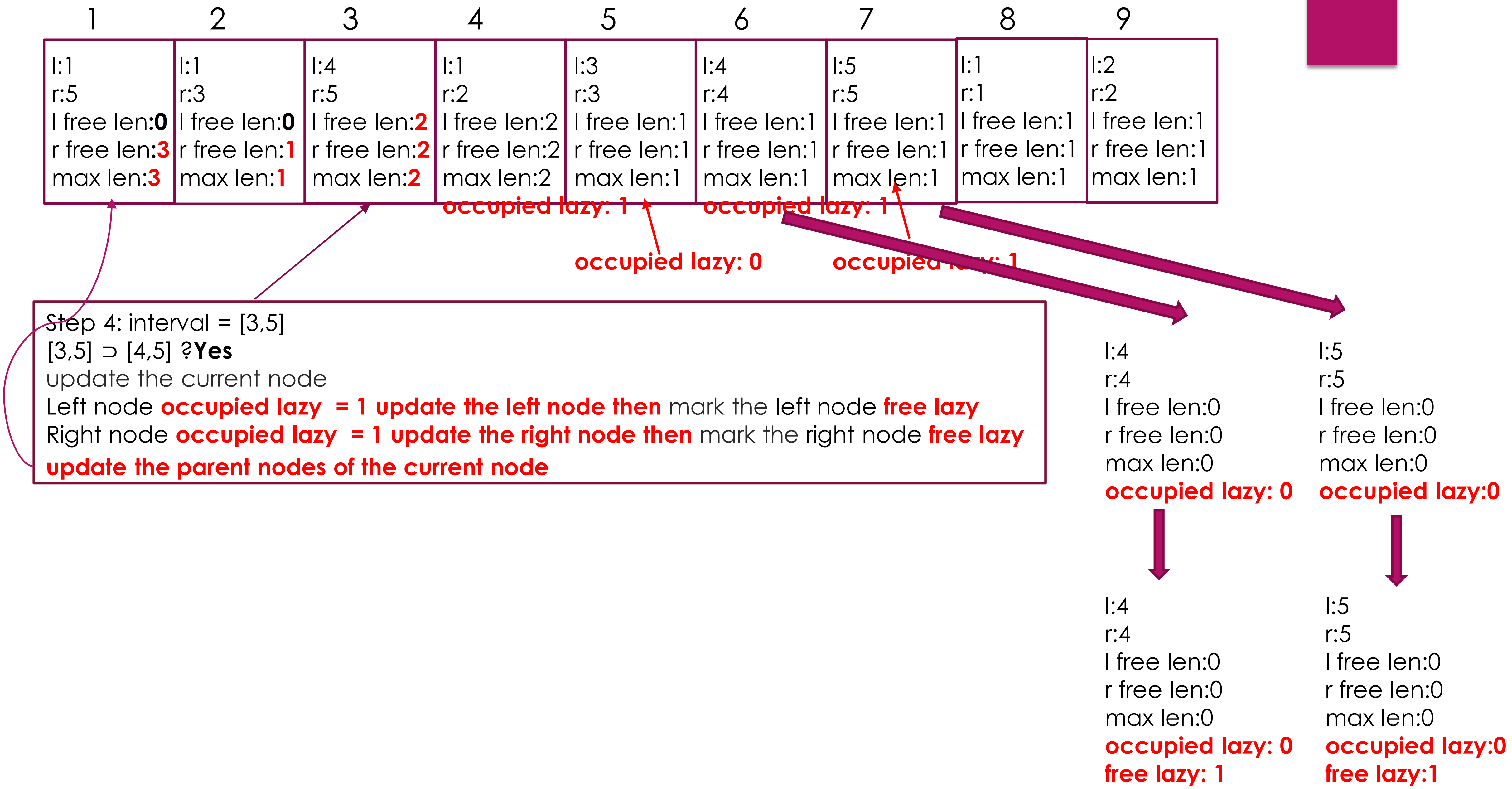
Step 2: interval = [3,5]
[3,5] ⊃ [1,3] ?No
Left node(**index:4**): [1,2] ∩ [3,5] = ∅ stop
Right node(**index:5**): [3] ∩ [3,5]≠∅ **occupied lazy = 1 update the right node then search right node**

Step 3: interval = [3,5]
[3,5] ⊃ [3,3] ?**Yes**
simply update the current node and mark the children free lazy
update the parent nodes of the current node

l:3
r:3
l free len:0
r free len:0
max len:0
occupied lazy:0

l:3
r:3
l free len:1
r free len:1
max len:1
occupied lazy:0

2 3 5 **check out room 3,4,5**



1	2	3	4	5	6	7	8	9
l:1	l:1	l:4	l:1	l:3	l:4	l:5	l:1	l:2
r:5	r:3	r:5	r:2	r:3	r:4	r:5	r:1	r:2
l free len:**0**	l free len:**0**	l free len:**2**	l free len:2	l free len:1	l free len:1	l free len:1	l free len:1	l free len:1
r free len:**3**	r free len:**1**	r free len:**2**	r free len:2	r free len:1	r free len:1	r free len:1	r free len:1	r free len:1
max len:**3**	max len:**1**	max len:**2**	max len:2	max len:1	max len:1	max len:1	max len:1	max len:1

occupied lazy: 1

occupied lazy: 1

occupied lazy: 0

occupied lazy: 1

Step 4: interval = [3,5]
[3,5] ⊃ [4,5] ?**Yes**
update the current node
Left node **occupied lazy = 1 update the left node then** mark the left node **free lazy**
Right node **occupied lazy = 1 update the right node then** mark the right node **free lazy**
update the parent nodes of the current node

l:4
r:4
l free len:0
r free len:0
max len:0
occupied lazy: 0

l:5
r:5
l free len:0
r free len:0
max len:0
occupied lazy:0

l:4
r:4
l free len:0
r free len:0
max len:0
occupied lazy: 0
free lazy: 1

l:5
r:5
l free len:0
r free len:0
max len:0
occupied lazy:0
free lazy:1