

# Προχωρημένες Δομές Δεδομένων

Ιούνιος 2012

Γιάννης Χατζημίκος  
[yannis@chatzimichos.gr](mailto:yannis@chatzimichos.gr)

Προκατασκευαστικό Camp 24ου ΠΔΠ

# Προχωρημένες Δομές Δεδομένων

1) **Union-Find (Disjoint sets)**

2) **Hash tables** -- Βαλκανιάδες

3) **Tries**

4) Δέντρα

- Interval Trees
- **Binary Indexed Trees**
- Quad Trees -- Βαλκανιάδες

5) **Suffix Arrays** -- Βαλκανιάδες

# Tries

- Είναι μια δεντρική δομή δεδομένων που μας επιτρέπει να αποθηκεύουμε σύνολα συμβολοσειρών και να εκτελούμε πράξεις σε αυτά.
- Μπορούμε να εκτελούμε πράξεις on-line (δεν χρειάζεται να έχουμε όλα τα δεδομένα από πριν).
- Χρησιμοποιούνται συχνά για αναπαράσταση λεξικών.

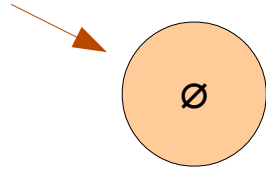
# Tries: Πράξεις

- Προσθήκη συμβολοσειράς
- Αφαίρεση συμβολοσειράς
- Έλεγχος για ύπαρξη συμβολοσειράς
- Αναζήτηση/καταμέτρηση συμβολοσειρών με το ίδιο πρόθεμα
- Αλφαβητική ταξινόμηση συμβολοσειρών
  - Εύρεση κ-οστής λέξης
- ...

# Tries: Κατασκευή

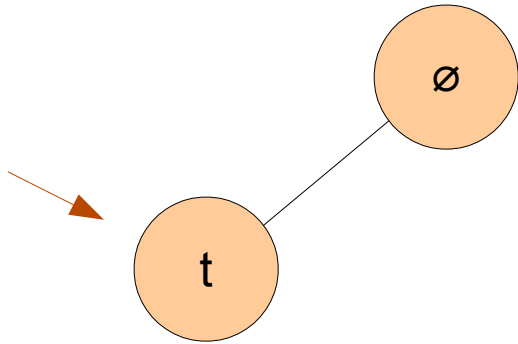
- Αρχικά έχουμε μια “ψεύτικη” ρίζα που αντιστοιχεί στην κενή συμβολοσειρά.
- Καθένας από τους υπόλοιπους κόμβους αντιστοιχεί σε κάποιο γράμμα της αλφαβήτου.
- Για να προσθέσουμε μια λέξη ξεκινάμε από την ρίζα και ακολουθούμε το μονοπάτι που δείχνουν τα γράμματά της, προσθέτοντας όσους κόμβους δεν υπάρχουν.

# Tries: Παράδειγμα κατασκευής



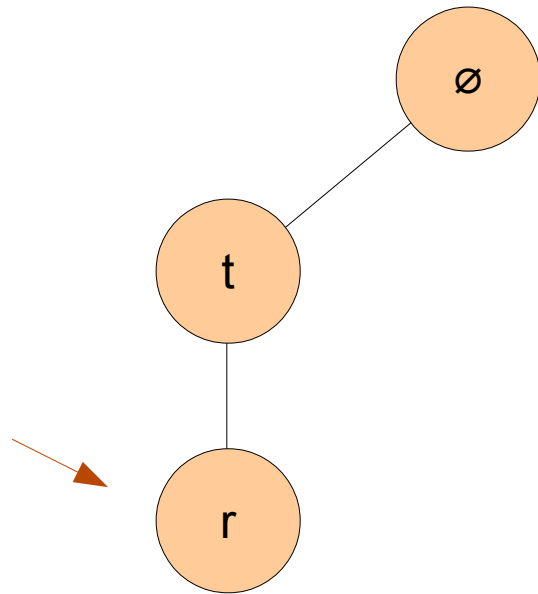
→ add “tree”

# Tries: Παράδειγμα κατασκευής



→ add “tree”

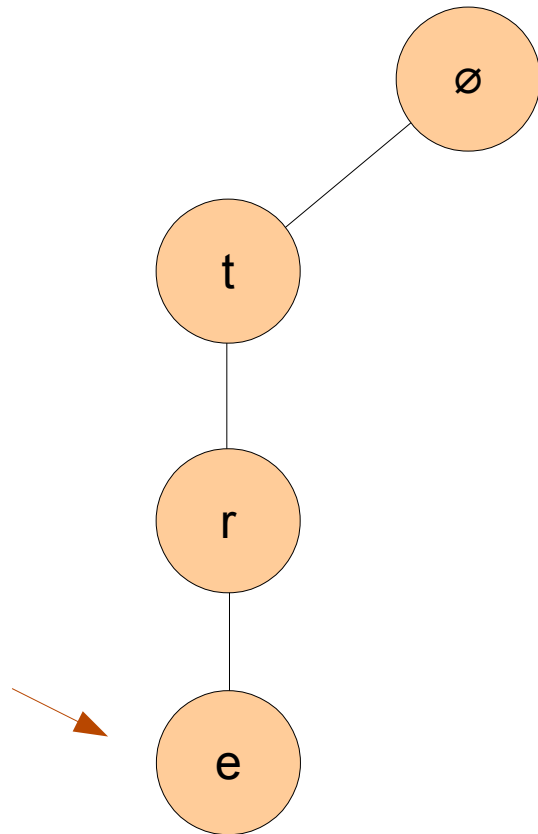
# Tries: Παράδειγμα κατασκευής



→ add “tree”

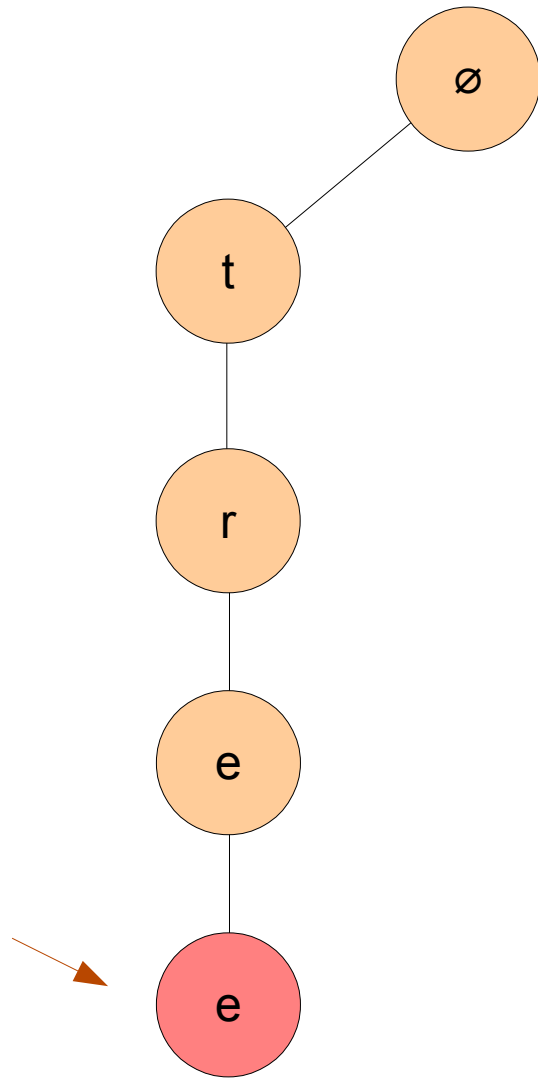


# Tries: Παράδειγμα κατασκευής



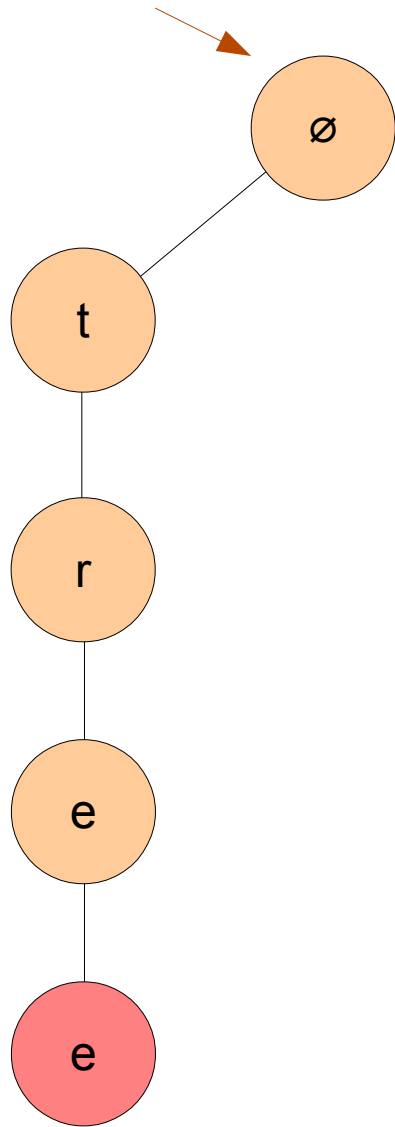
→ add "tree"

# Tries: Παράδειγμα κατασκευής



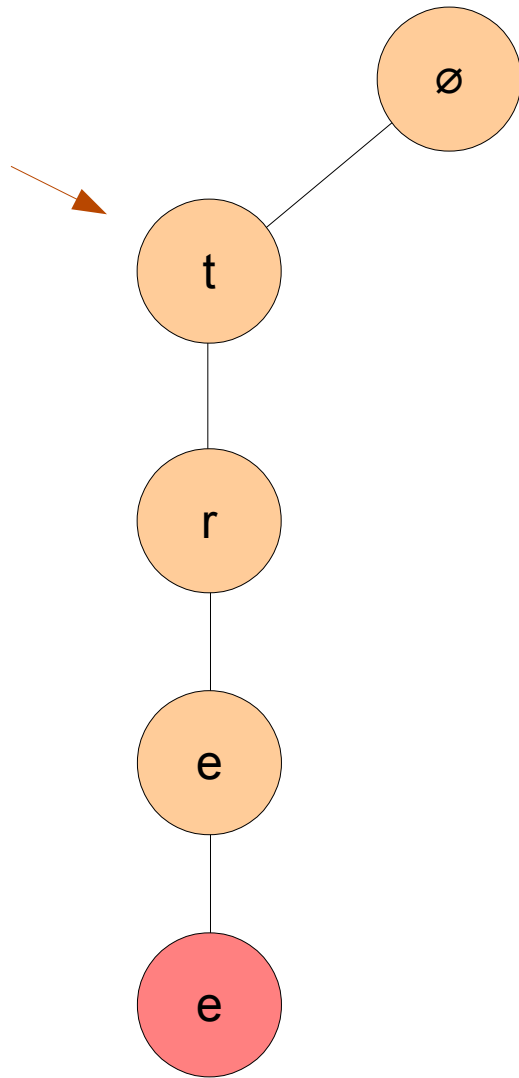
→ add “tree”

# Tries: Παράδειγμα κατασκευής



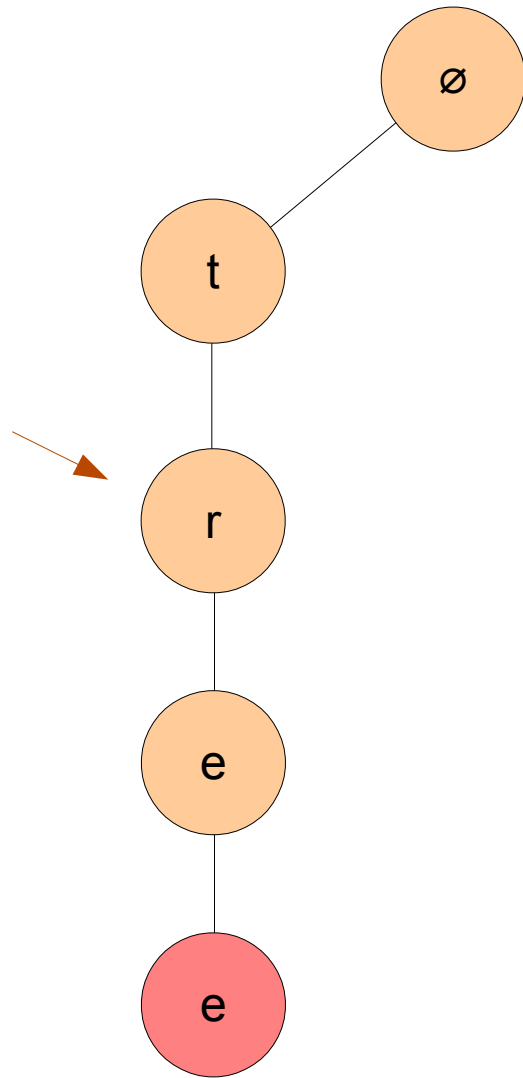
add "tree"  
→ add "trie"

# Tries: Παράδειγμα κατασκευής



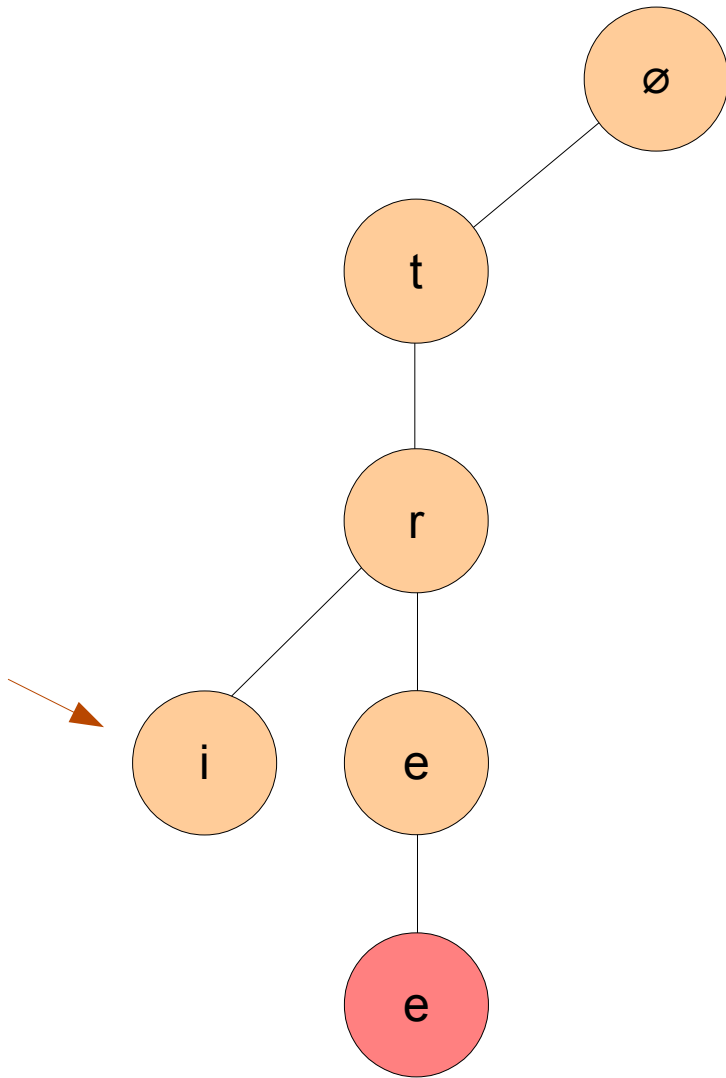
add "tree"  
→ add "trie"

# Tries: Παράδειγμα κατασκευής



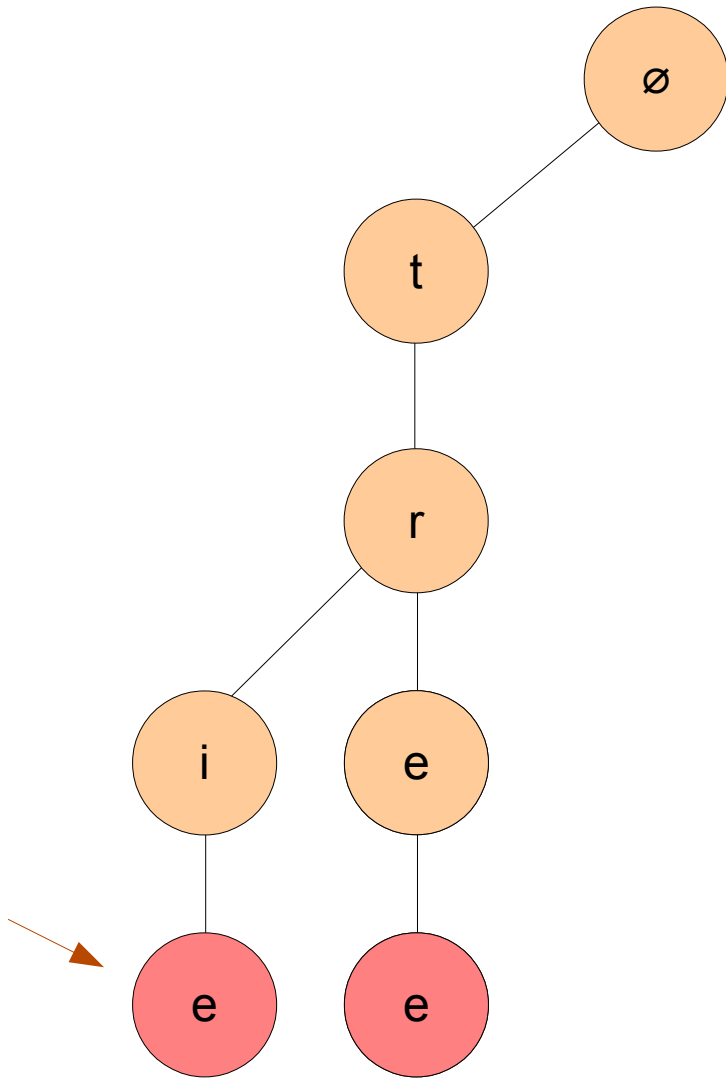
add "tree"  
→ add "trie"

# Tries: Παράδειγμα κατασκευής



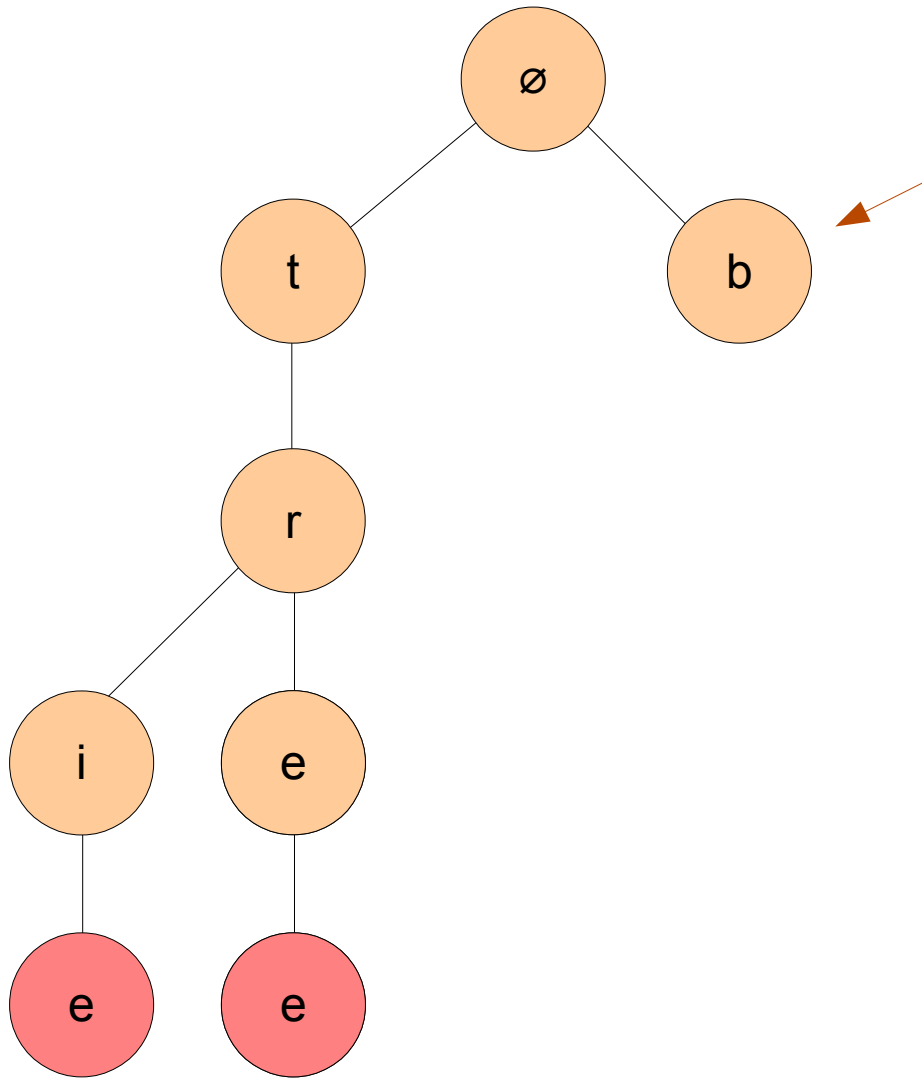
add "tree"  
→ add "trie"

# Tries: Παράδειγμα κατασκευής



add "tree"  
→ add "trie"

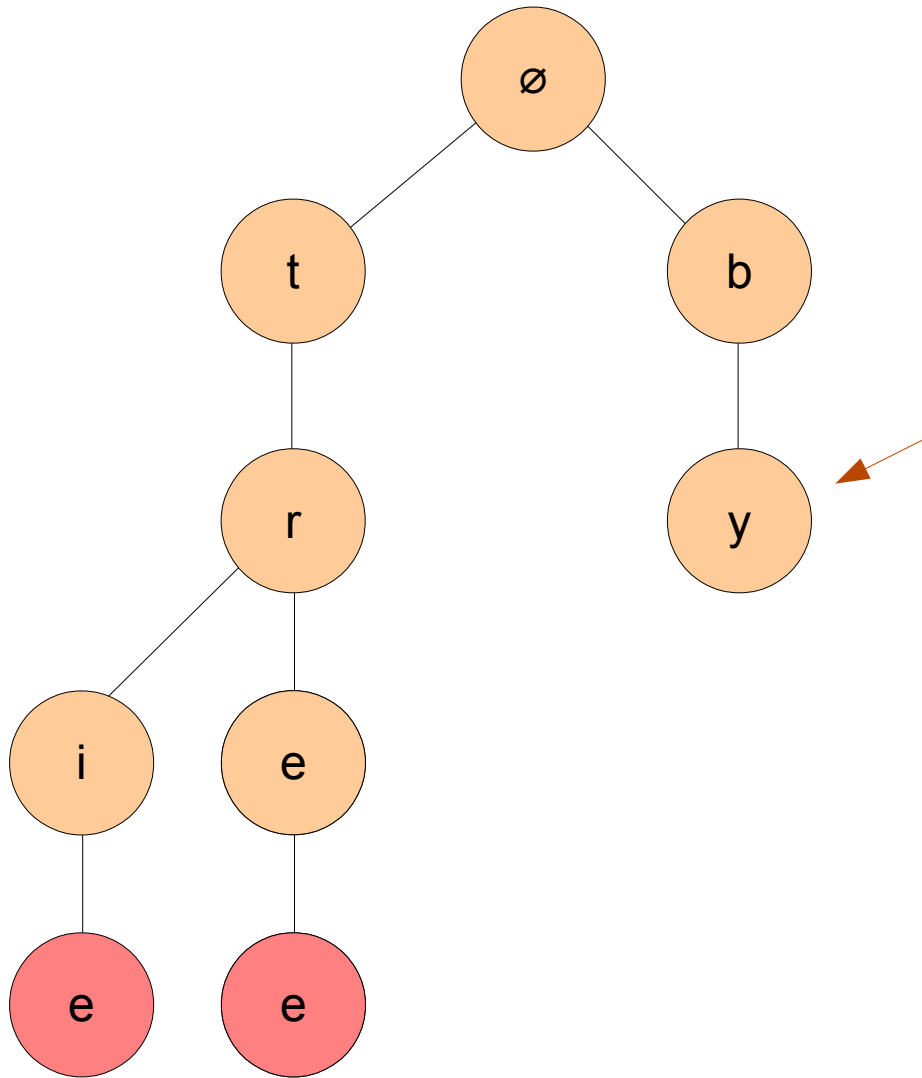
# Tries: Παράδειγμα κατασκευής



add "tree"  
add "trie"  
→ add "bye"

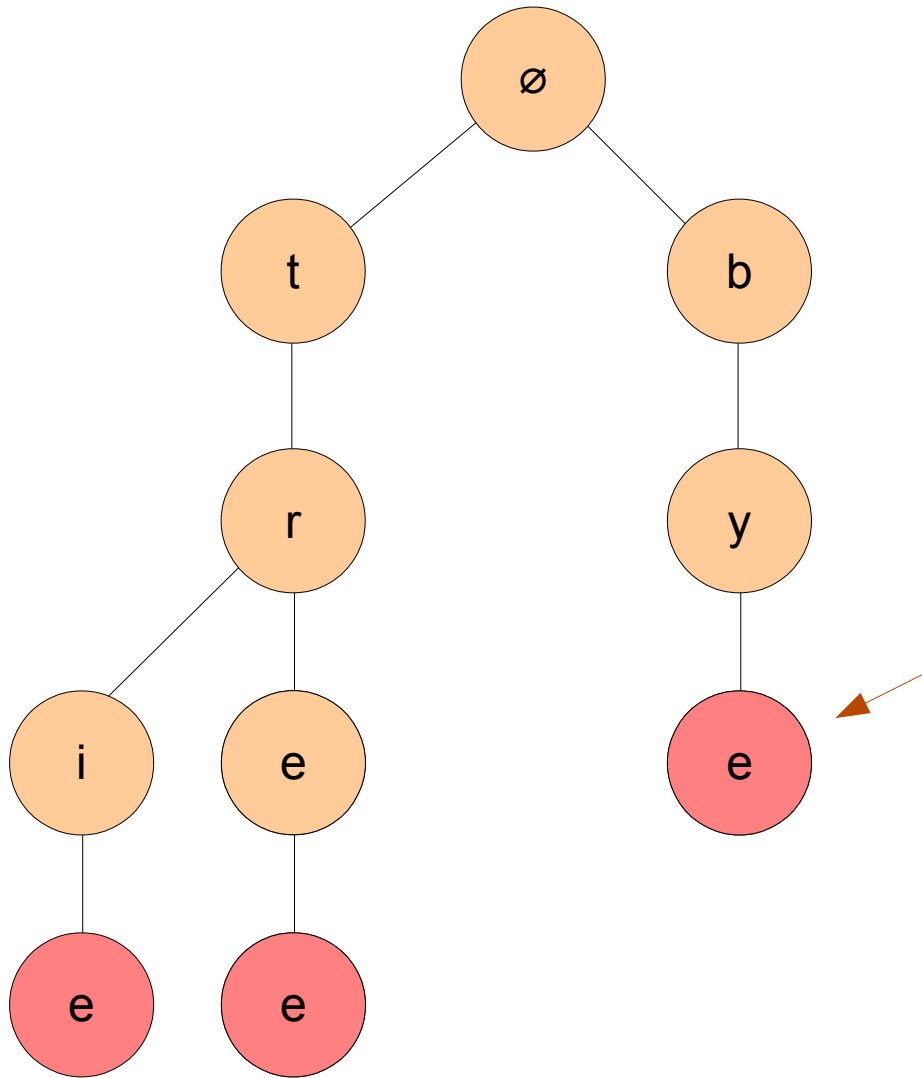


# Tries: Παράδειγμα κατασκευής



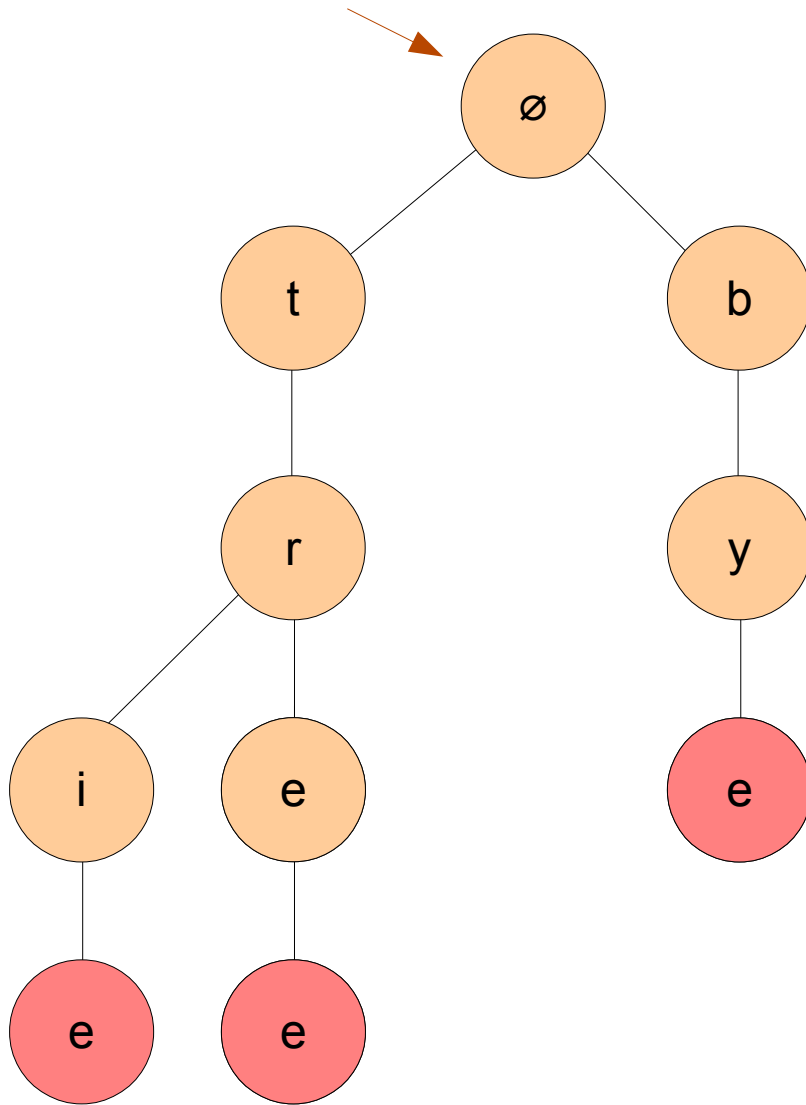
add "tree"  
add "trie"  
→ add "bye"

# Tries: Παράδειγμα κατασκευής



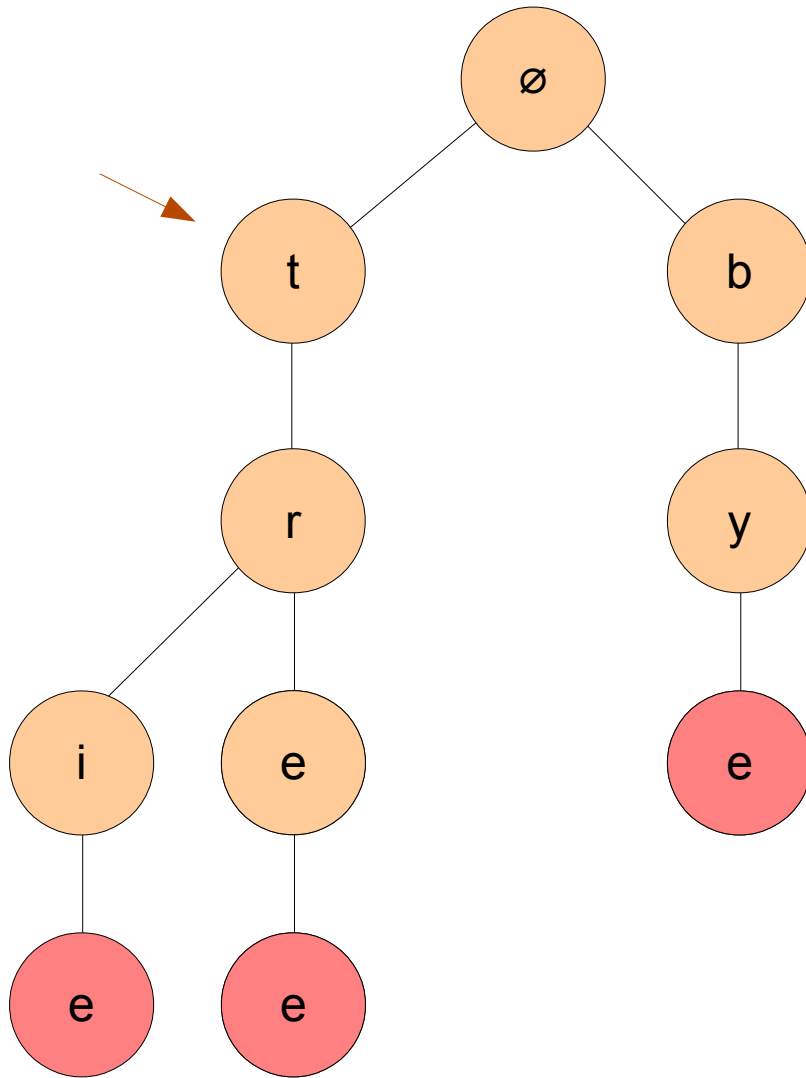
add "tree"  
add "trie"  
→ add "bye"

# Tries: Παράδειγμα κατασκευής



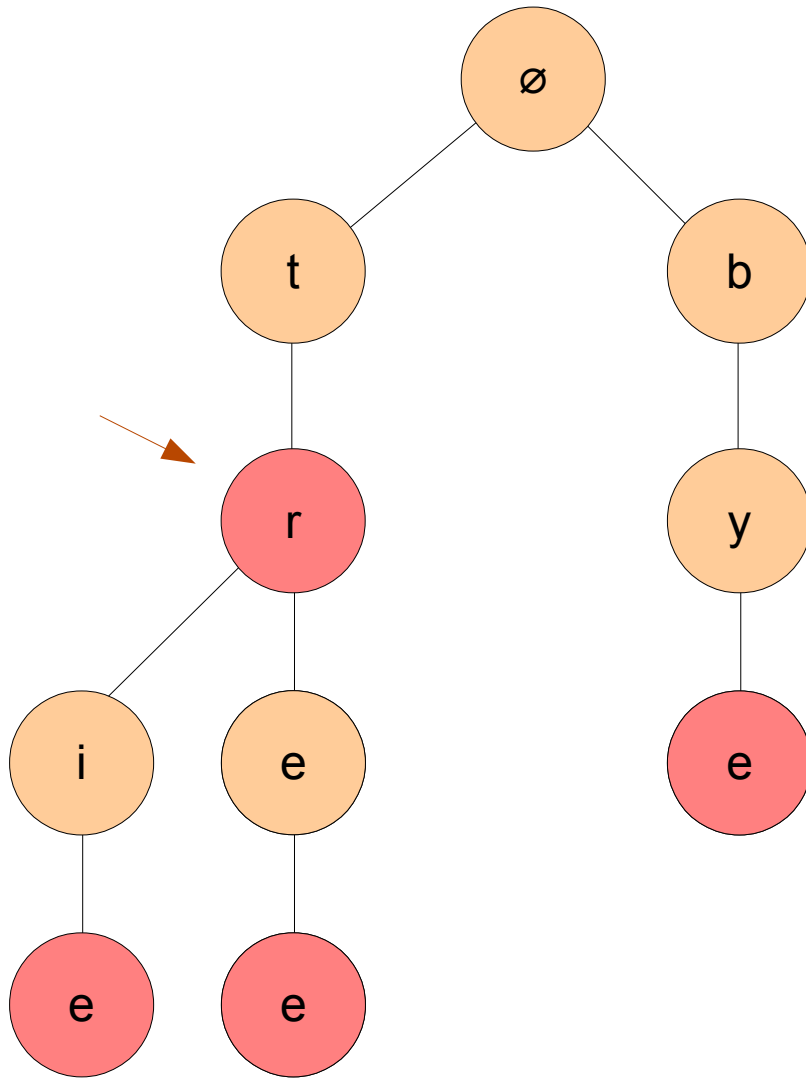
add "tree"  
add "trie"  
add "bye"  
→ add "tr"

# Tries: Παράδειγμα κατασκευής



add "tree"  
add "trie"  
add "bye"  
→ add "tr"

# Tries: Παράδειγμα κατασκευής



add "tree"  
add "trie"  
add "bye"  
→ add "tr"

# Tries: Υλοποίηση (the IOI way)

```
#define N 50000

struct node {
    int children[26];
    char isWord;
};

struct node Trie[N];
int trieNodeCount;

int initialize() {
    trieNodeCount = 1; /* προσθέτουμε την “ψεύτικη” ρίζα */
}
```

# Tries: Υλοποίηση της add

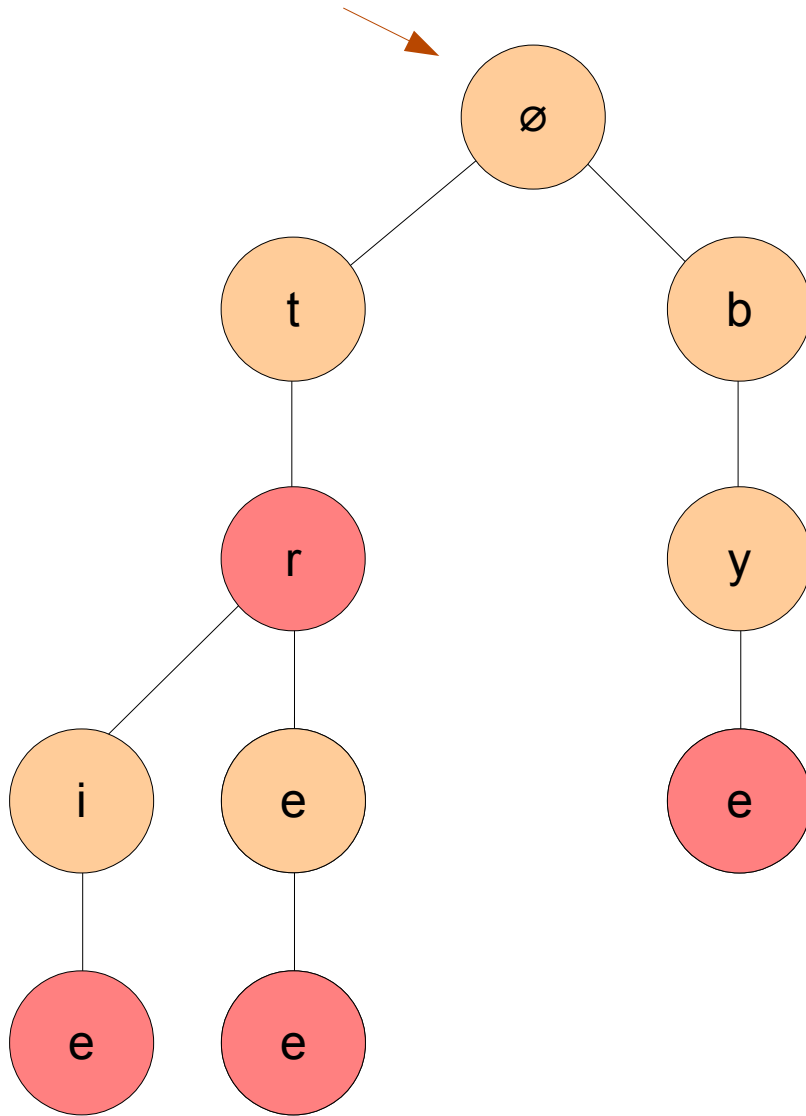
```
int add(char *word, int length) {
    int i, nextNode, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα */

    for (i=0; i<length; i++) {
        nextNode = Trie[curNode].children[ word[i] - 'a' ];

        if ( nextNode == 0 ) { /* αν δεν υπάρχει ο κόμβος στο
                                trie, τότε τον δημιουργούμε */

            trieNodeCount++;
            Trie[curNode].children[ word[i] - 'a' ] = trieNodeCount;
            curNode = trieNodeCount;
        }
        else {
            curNode = nextNode;
        }
    }
    Trie[curNode].isWord = 1;
}
```

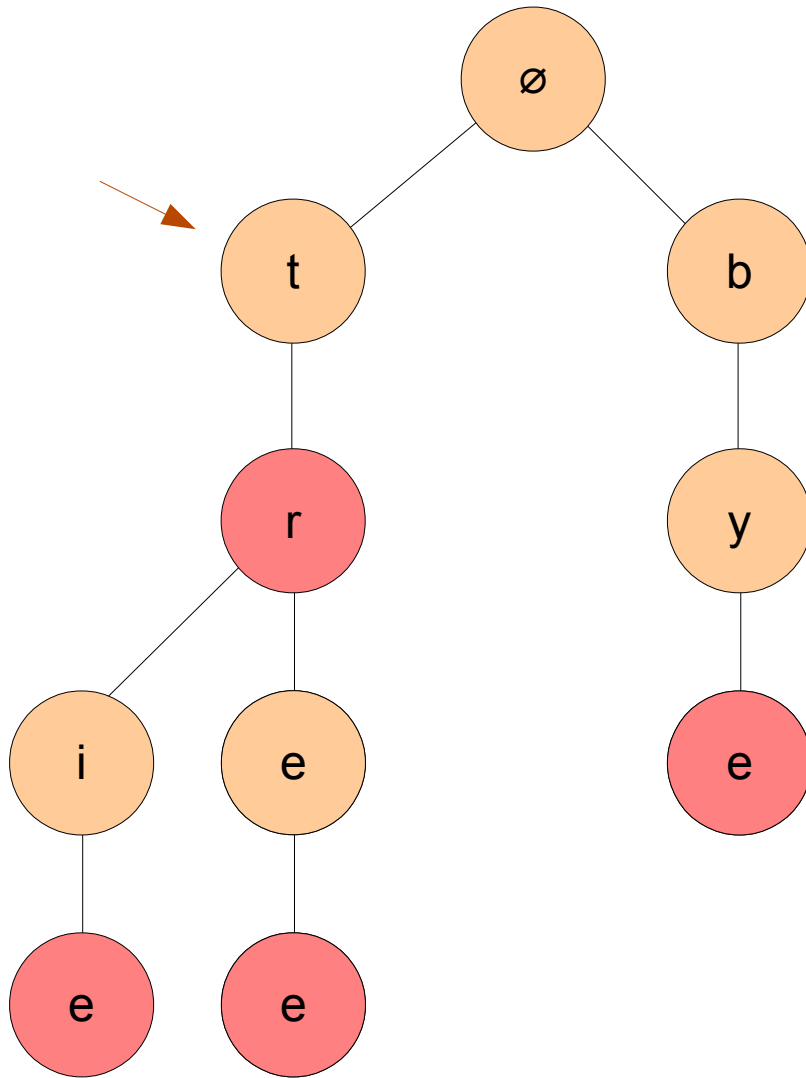
# Tries: Παραδείγματα



add "tree"  
add "trie"  
add "bye"  
add "tr"  
→ remove "trie"

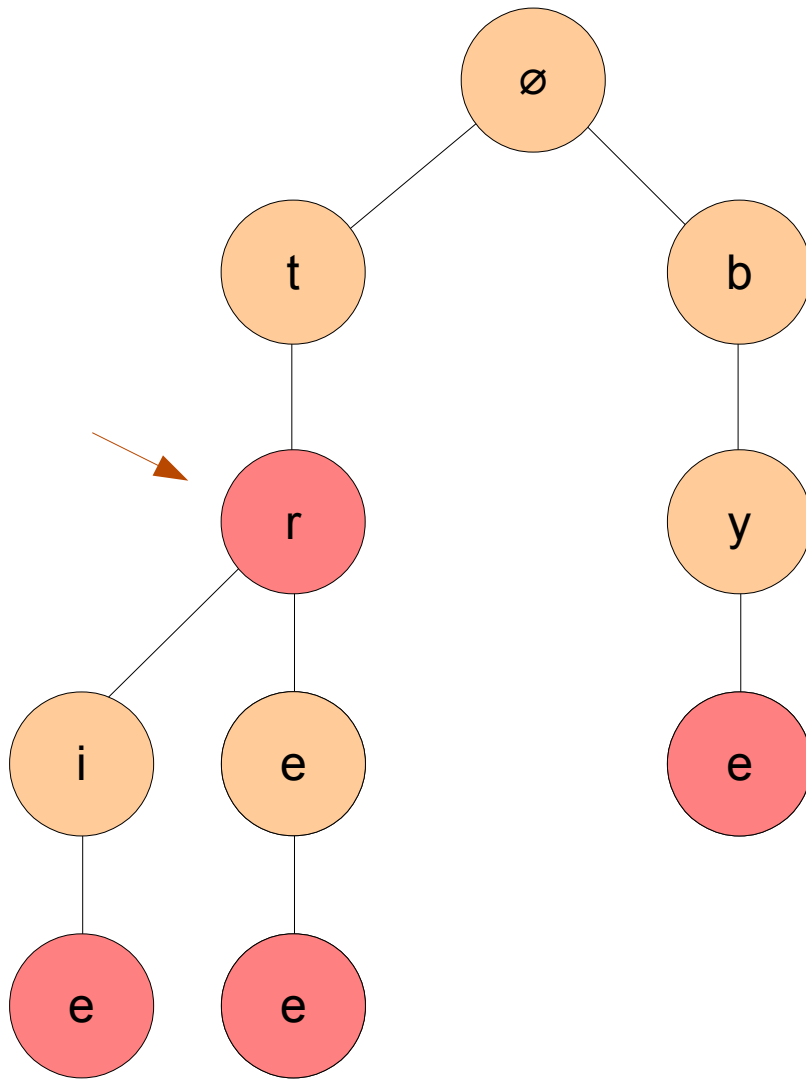


# Tries: Παραδείγματα



add "tree"  
add "trie"  
add "bye"  
add "tr"  
→ remove "trie"

# Tries: Παραδείγματα



add "tree"

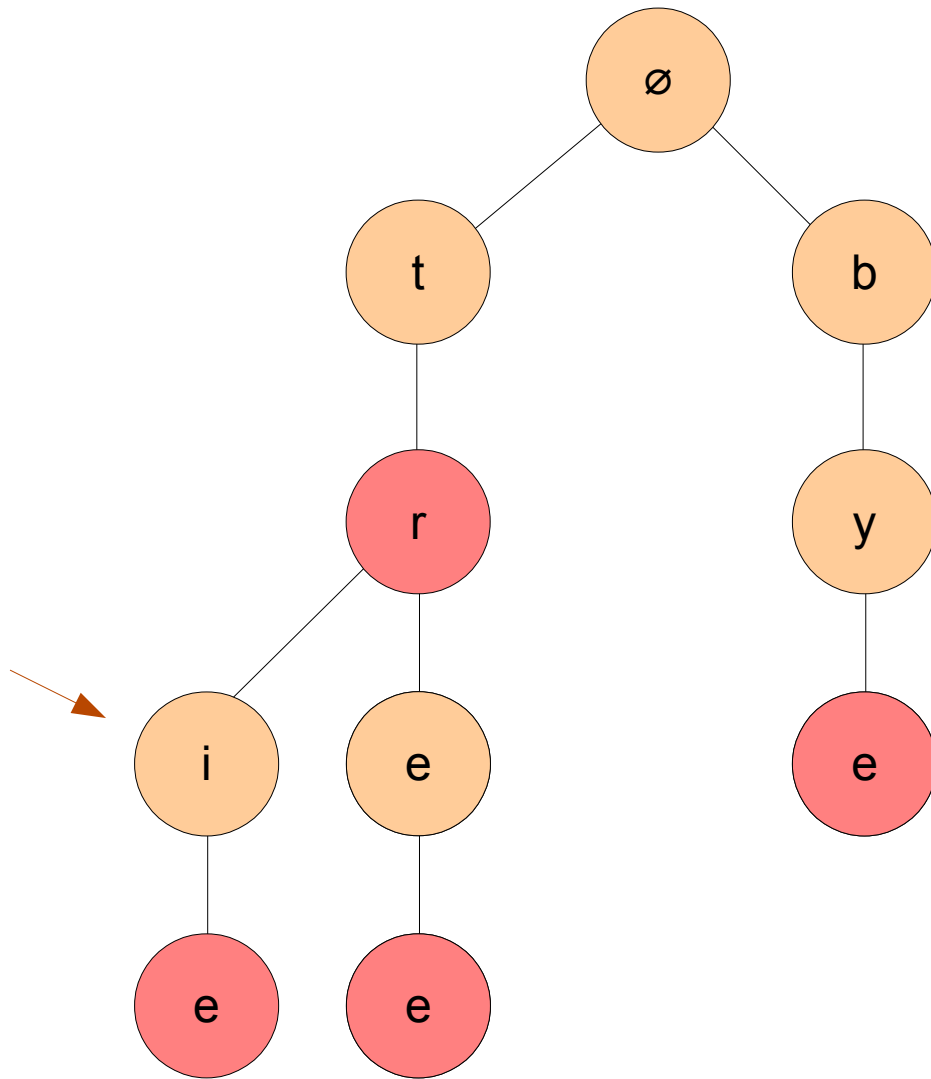
add "trie"

add "bye"

add "tr"

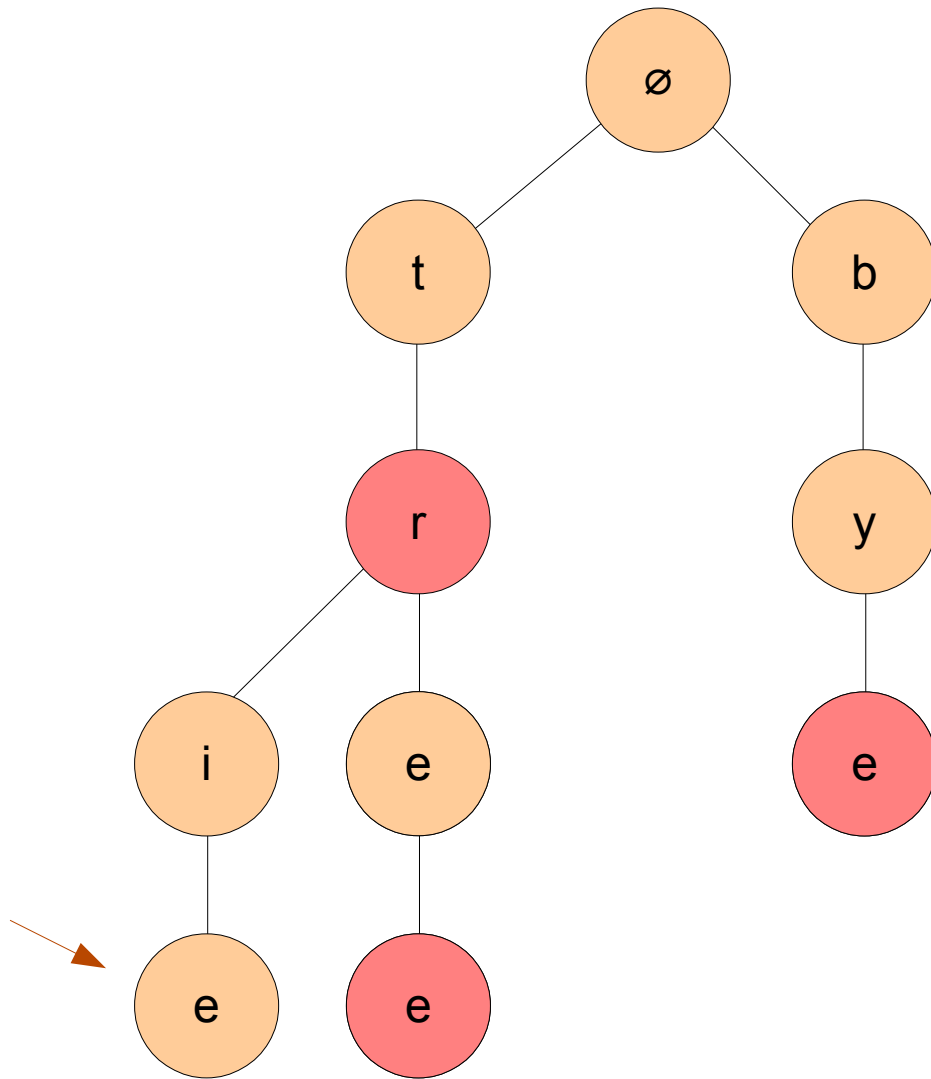
→ remove "trie"

# Tries: Παραδείγματα



add "tree"  
add "trie"  
add "bye"  
add "tr"  
→ remove "trie"

# Tries: Παραδείγματα



αλλάζουμε το isWord σε 0

add "tree"  
add "trie"  
add "bye"  
add "tr"  
→ remove "trie"

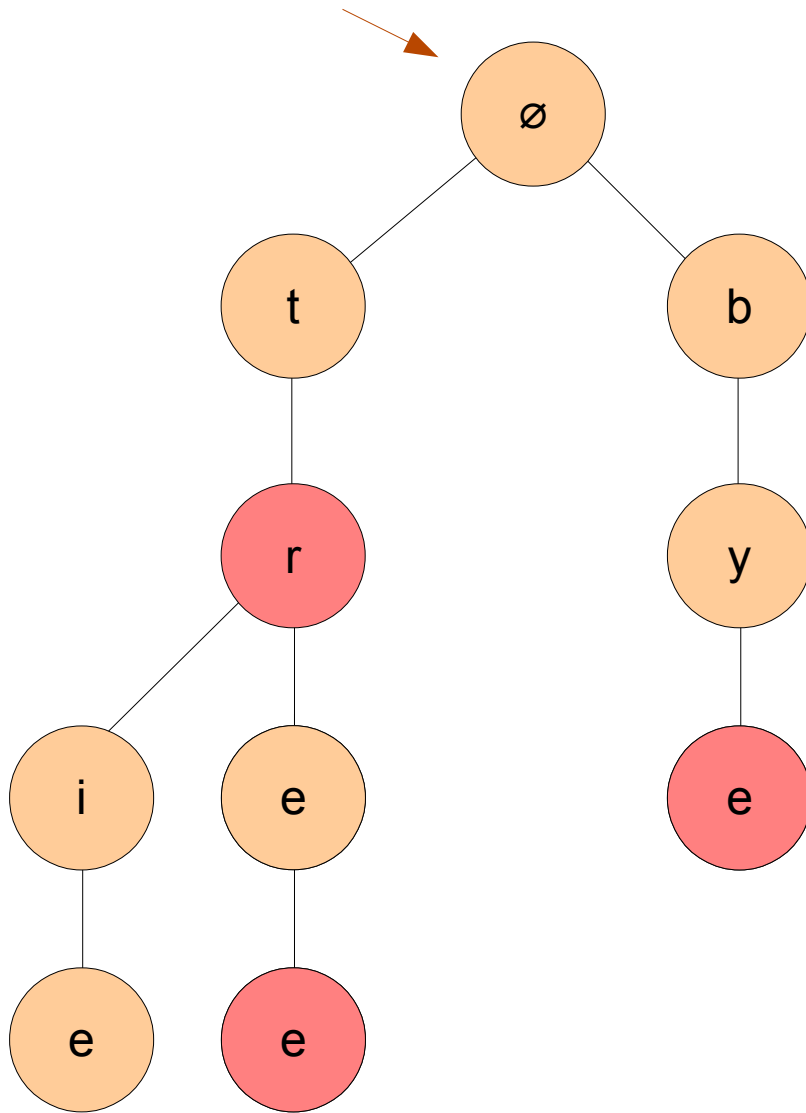
# Tries: Υλοποίηση της remove

```
int remove(char word, int length) { /* ακολουθούμε το μονοπάτι μέχρι τον
                                     κόμβο του τελευταίου γράμματος και στη
                                     συνέχεια αλλάζουμε το isWord σε 0 */

    int i, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα του trie */
    for (i=0; i<length; i++) {
        curNode = Trie[curNode].children[ word[i] - 'a' ];

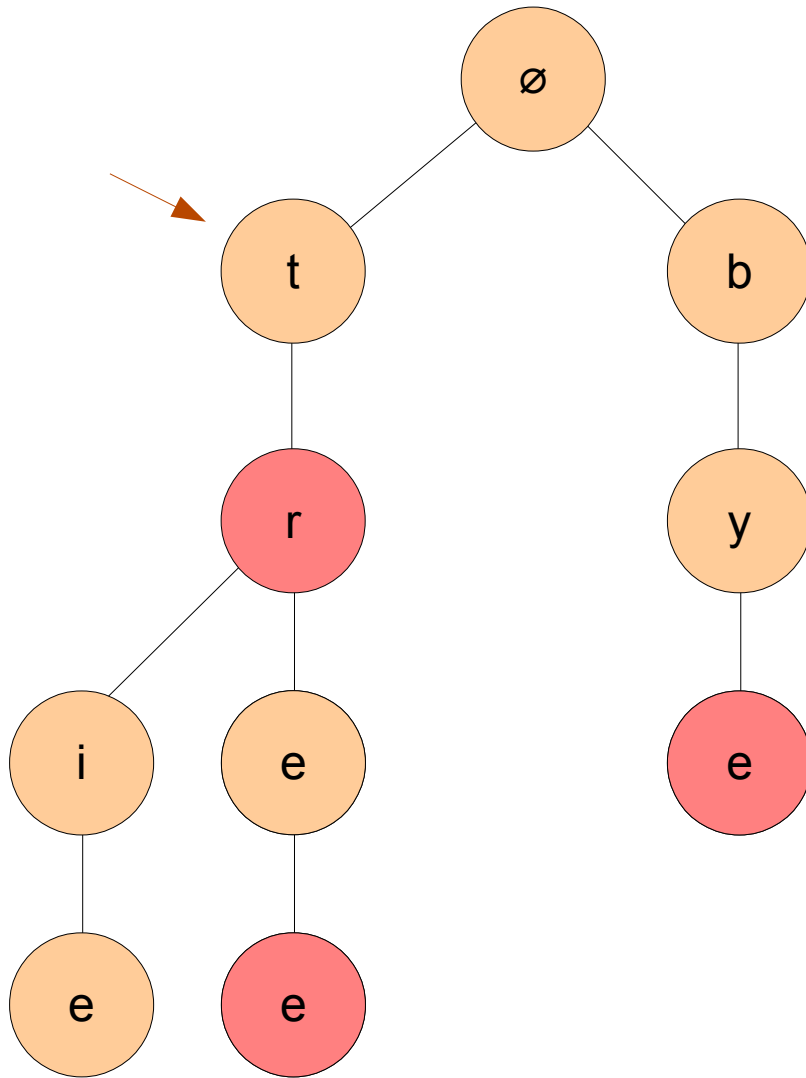
        assert(curNode != 0); /* υποθέτουμε ότι η λέξη που θέλουμε να
                               διαγράψουμε υπάρχει πάντα */
    }
    Trie[curNode].isWord = 0;
}
```

# Tries: Παραδείγματα



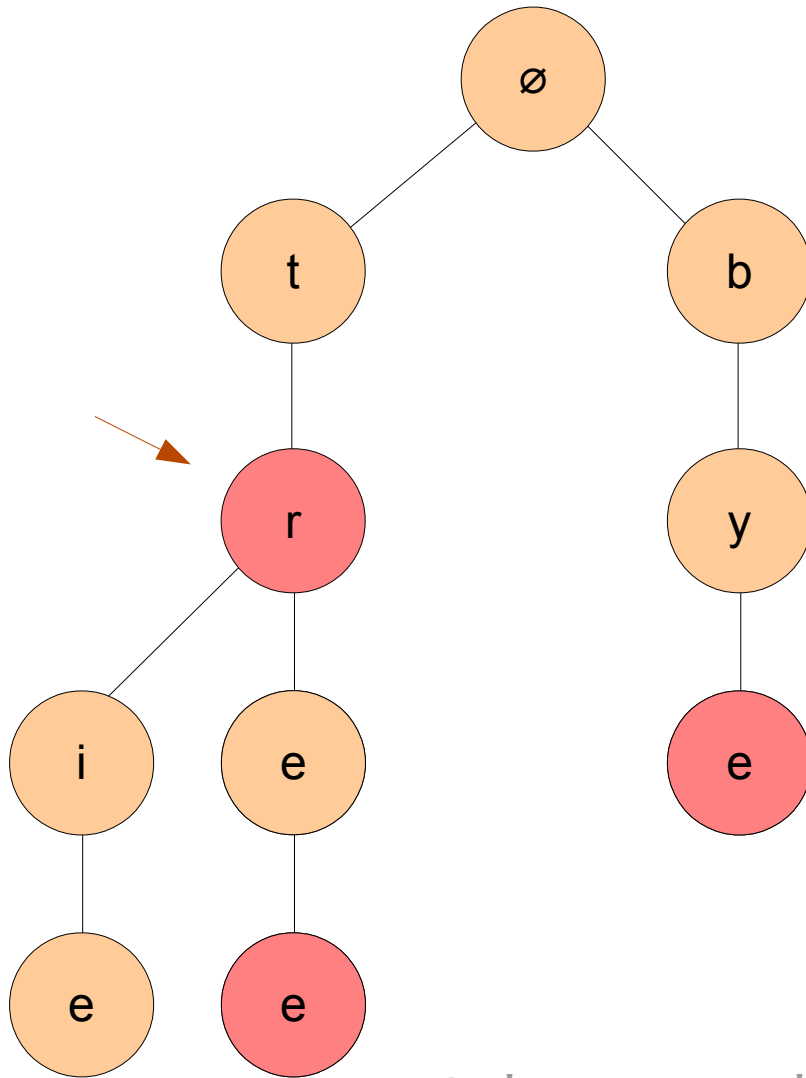
add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
→ check "tr"

# Tries: Παραδείγματα



add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
→ check "tr"

# Tries: Παραδείγματα

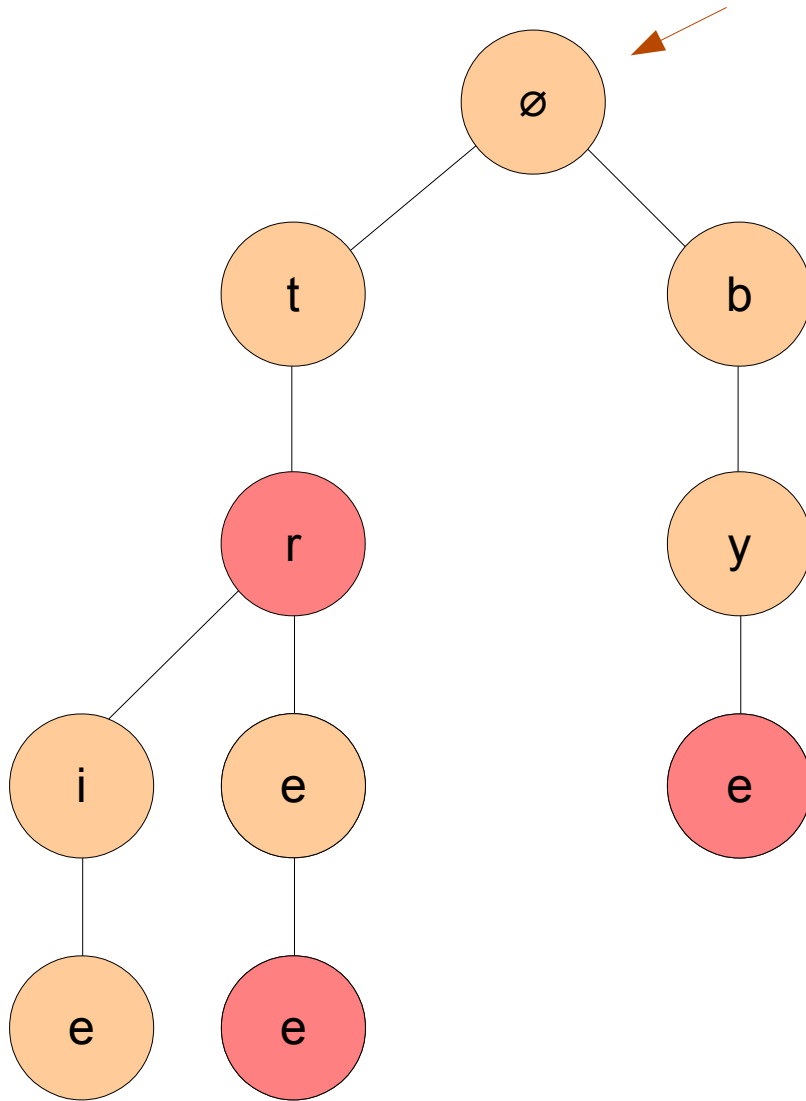


add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
→ check "tr" true

Φτάσαμε στο κόμβο του τελευταίου γράμματος, ελέγχουμε αν το isWord αυτού του κόμβου είναι 1

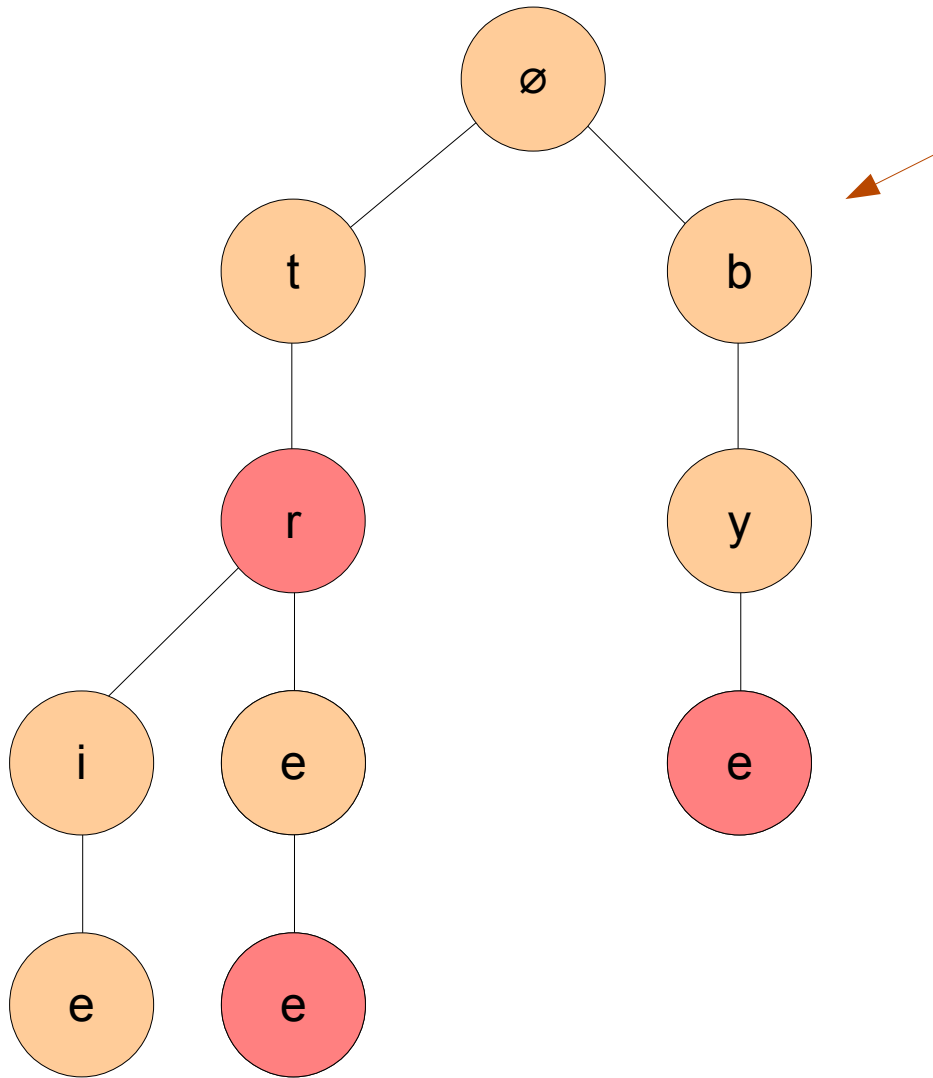


# Tries: Παραδείγματα



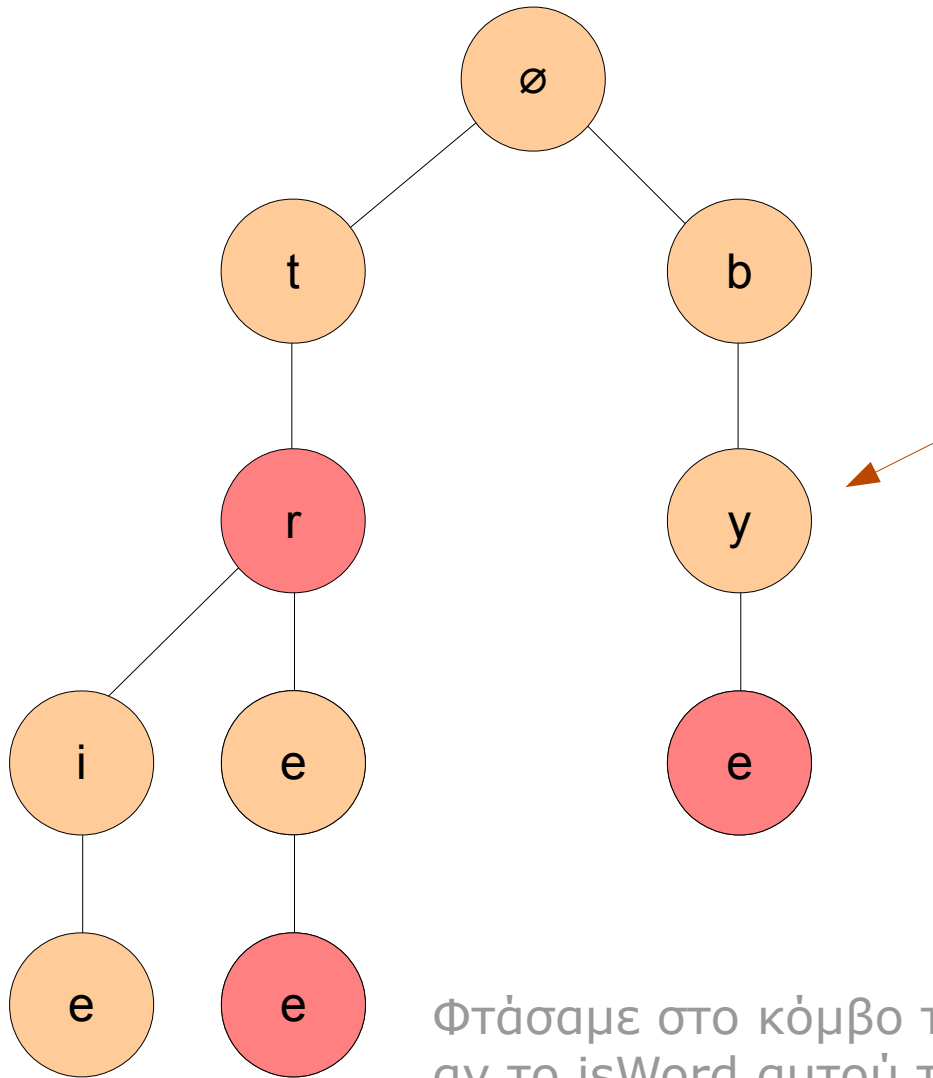
add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
check "tr" true  
→ check "by"

# Tries: Παραδείγματα



add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
check "tr" true  
→ check "by"

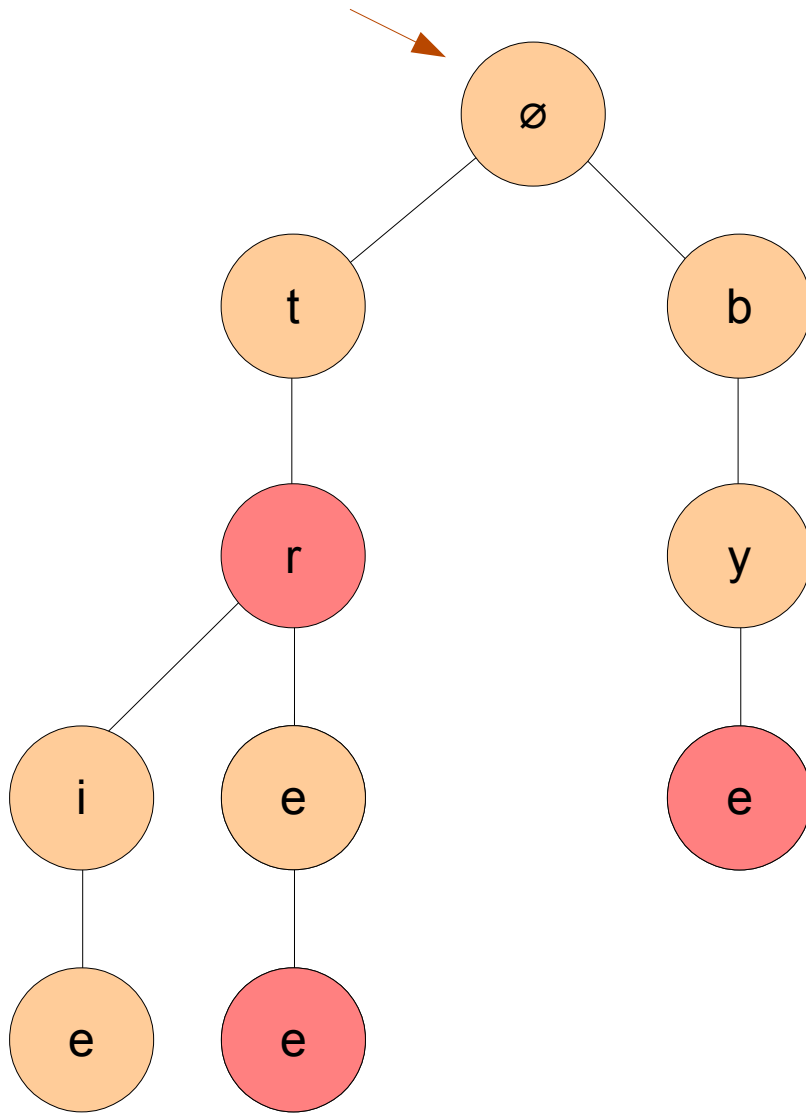
# Tries: Παραδείγματα



add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
check "tr" true  
→ check "by" false

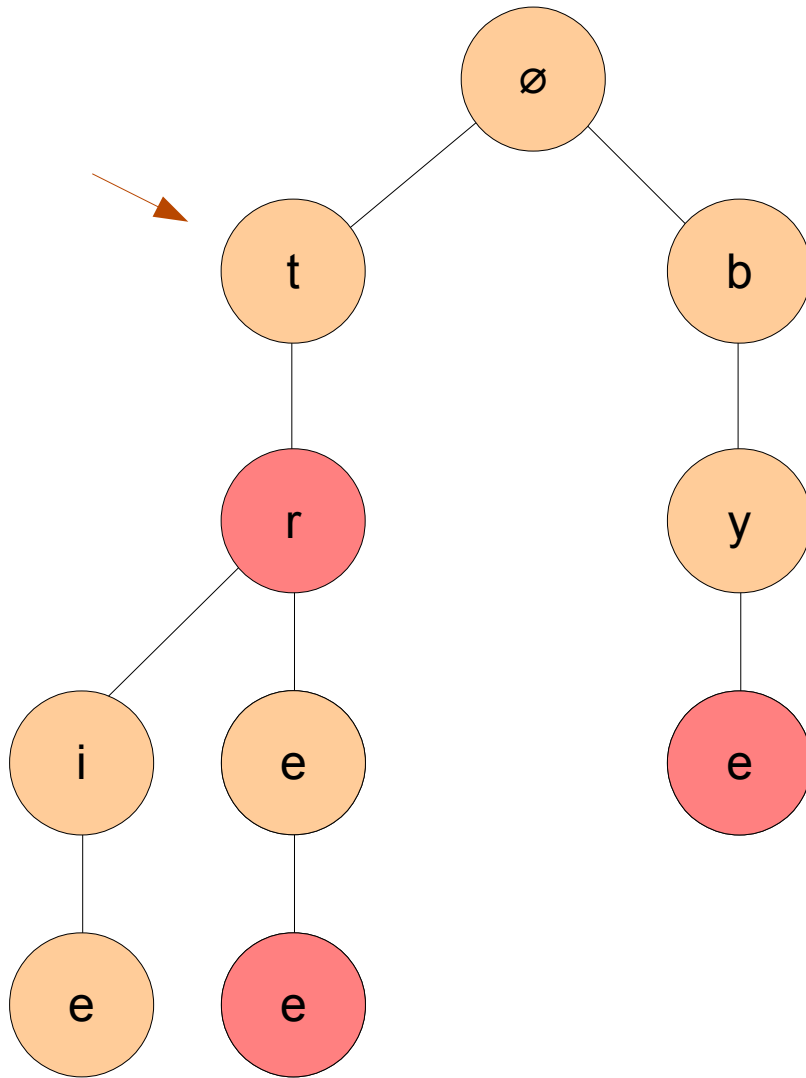
Φτάσαμε στο κόμβο του τελευταίου γράμματος, ελέγχουμε αν το isWord αυτού του κόμβου είναι 1

# Tries: Παραδείγματα



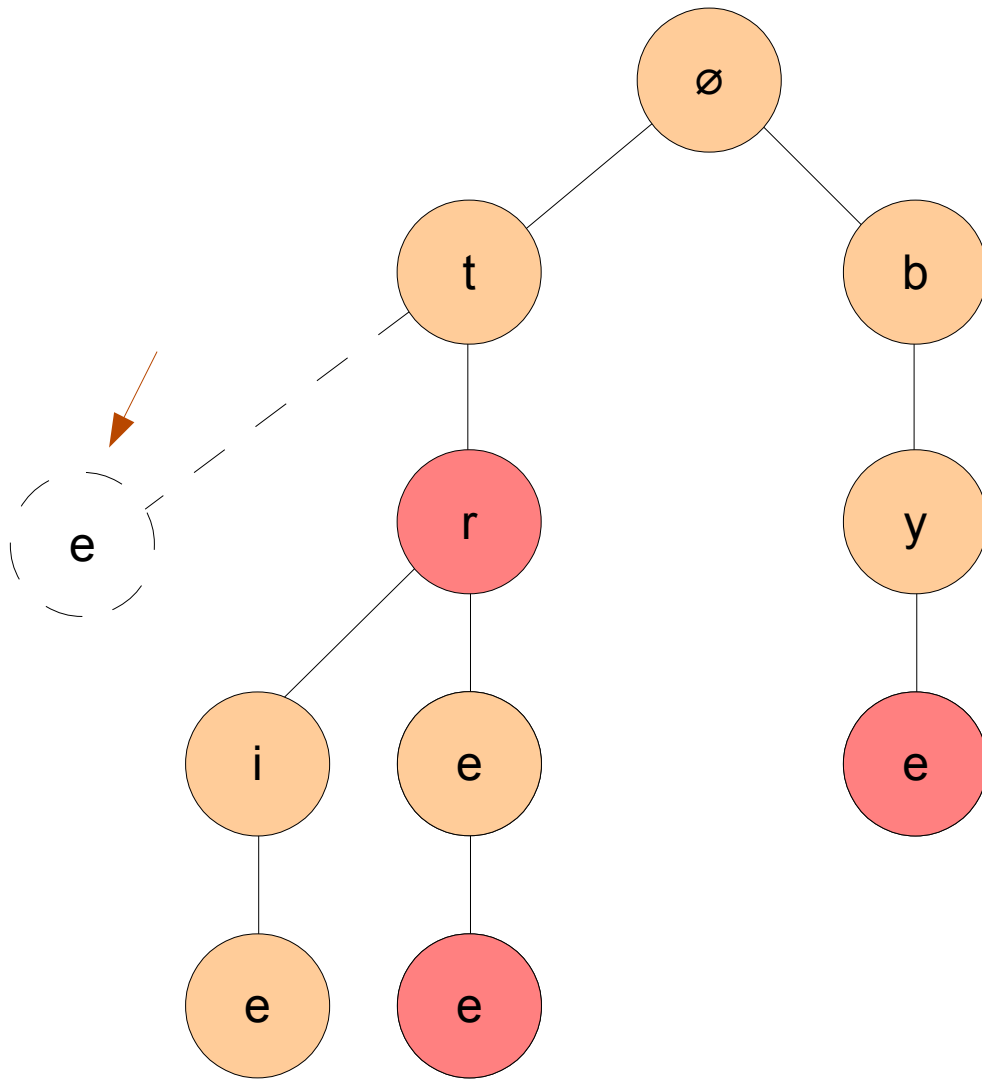
add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
check "tr" true  
check "by" false  
→ check "ten"

# Tries: Παραδείγματα



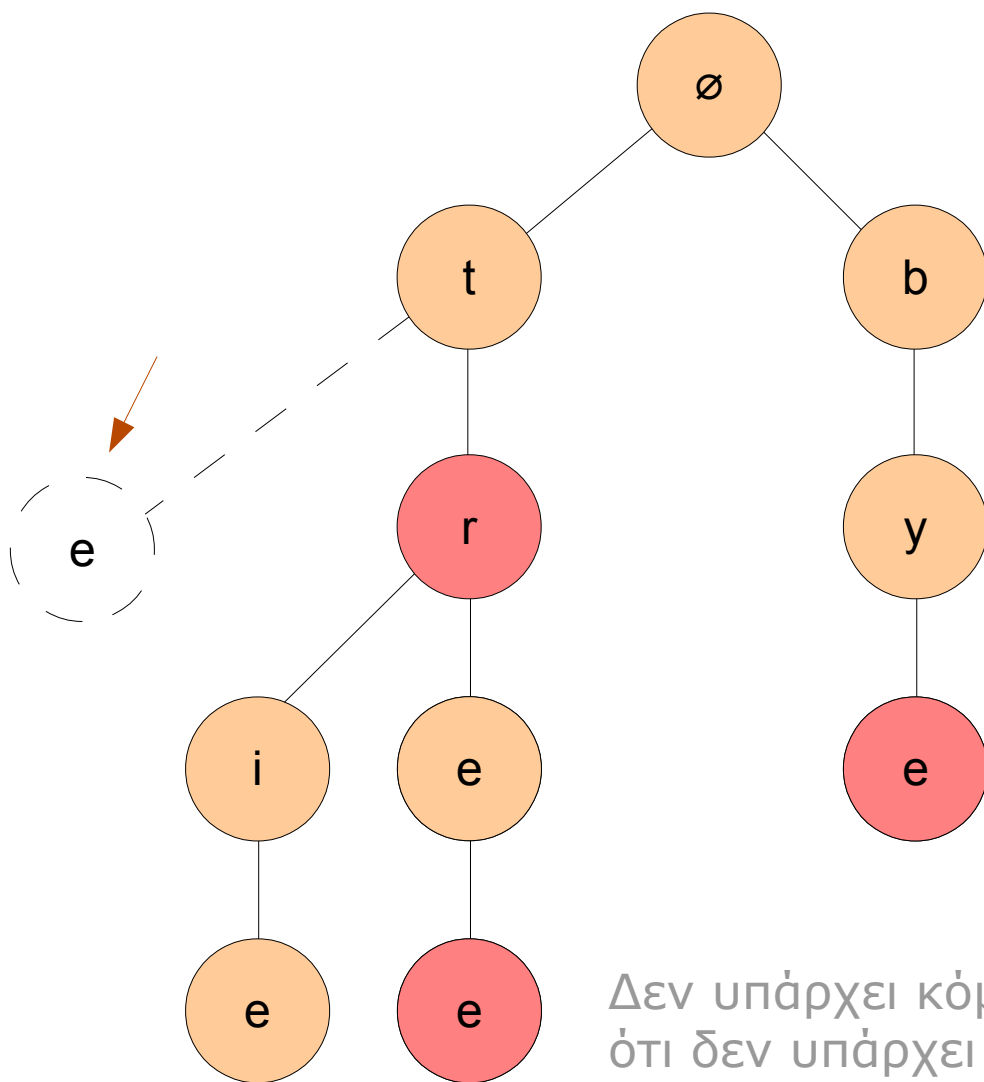
add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
check "tr" true  
check "by" false  
→ check "ten"

# Tries: Παραδείγματα



add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
check "tr" true  
check "by" false  
→ check "ten"

# Tries: Παραδείγματα



add "tree"  
add "trie"  
add "bye"  
add "tr"  
remove "trie"  
check "tr" true  
check "by" false  
→ check "ten" false

Δεν υπάρχει κόμβος προς το γράμμα "e". Αυτό σημαίνει ότι δεν υπάρχει λέξη που να αρχίζει από "te", άρα δεν υπάρχει και η συμβολοσειρά που ψάχνουμε. **Πρέπει να λάβουμε υπόψιν μας και αυτή την περίπτωση.**

# Tries: Υλοποίηση της check

```
bool check(char word, int length) {  
    int i, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα του trie */  
    for (i=0; i<length; i++) {  
        curNode = Trie[curNode].children[ word[i] - 'a' ];  
  
        if ( curNode == 0 ) { /* αν δεν υπάρχει ο κόμβος στο trie  
            τότε δεν υπάρχει και η λέξη που ψάχνουμε */  
            return false;  
        }  
    }  
    return (Trie[curNode].isWord == 1 ? true : false);  
}
```



# Tries: Παράδειγμα Επέκτασης (1)

- Δυνατότητα προσθήκης της ίδιας λέξης περισσότερες από μια φορές και
- αλλαγή της check ώστε να επιστρέφει πόσες φορές υπάρχει η λέξη στο trie (0 αν δεν υπάρχει).

# Tries: Υλοποίηση με wordCount

```
#define N 50000
```

```
struct node {
```

```
    int children[26];
```

```
    int wordCount;
```

```
};
```

```
struct node Trie[N];
```

```
int trieNodeCount;
```

```
int initialize() {
```

```
    trieNodeCount = 1; /* προσθέτουμε την “ψεύτικη” ρίζα */
```

```
}
```


Αλλαγή του isWord σε wordCount  
(από char γίνεται int).



# Tries: Υλοποίηση της add

```
int add(char *word, int length) {  
    int i, nextNode, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα */  
  
    for (i=0; i<length; i++) {  
        nextNode = Trie[curNode].children[ word[i] - 'a' ];  
  
        if ( nextNode == 0 ) { /* αν δεν υπάρχει ο κόμβος στο  
                                trie, τότε τον δημιουργούμε */  
  
            trieNodeCount++;  
            Trie[curNode].children[ word[i] - 'a' ] = trieNodeCount;  
            curNode = trieNodeCount;  
        }  
        else {  
            curNode = nextNode;  
        }  
    }  
    Trie[curNode].wordCount++;  
}
```

Αυξάνουμε το πλήθος των  
συμβολοσειρών που καταλήγουν σε  
αυτόν τον κόμβο κατά 1.




# Tries: Υλοποίηση της remove

```
int remove(char word, int length) { /* ακολουθούμε το μονοπάτι μέχρι τον
                                     κόμβο του τελευταίου γράμματος και στη
                                     συνέχεια αλλάζουμε το isWord σε 0 */

    int i, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα του trie */
    for (i=0; i<length; i++) {
        curNode = Trie[curNode].children[ word[i] - 'a' ];

        assert(curNode != 0); /* υποθέτουμε ότι η λέξη που θέλουμε να
                               διαγράψουμε υπάρχει πάντα */
    }
    Trie[curNode].wordCount--;
}
```



Μειώνουμε το πλήθος των  
συμβολοσειρών που καταλήγουν σε  
αυτόν τον κόμβο κατά 1.

# Tries: Υλοποίηση της check

Η συνάρτηση επιστρέφει  
πλέον αριθμό.

```
int check(char word, int length) {  
    int i, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα του trie */  
    for (i=0; i<length; i++) {  
        curNode = Trie[curNode].children[ word[i] - 'a' ];  
  
        if ( curNode == 0 ) { /* αν δεν υπάρχει ο κόμβος στο trie  
                               τότε δεν υπάρχει και η λέξη που ψάχνουμε */  
            return 0; ← Δεν υπάρχει μονοπάτι, άρα 0 λέξεις  
        }  
    }  
    return Trie[curNode].wordCount; ← Επιστρέφουμε το πλήθος  
    }                                  των λέξεων
```

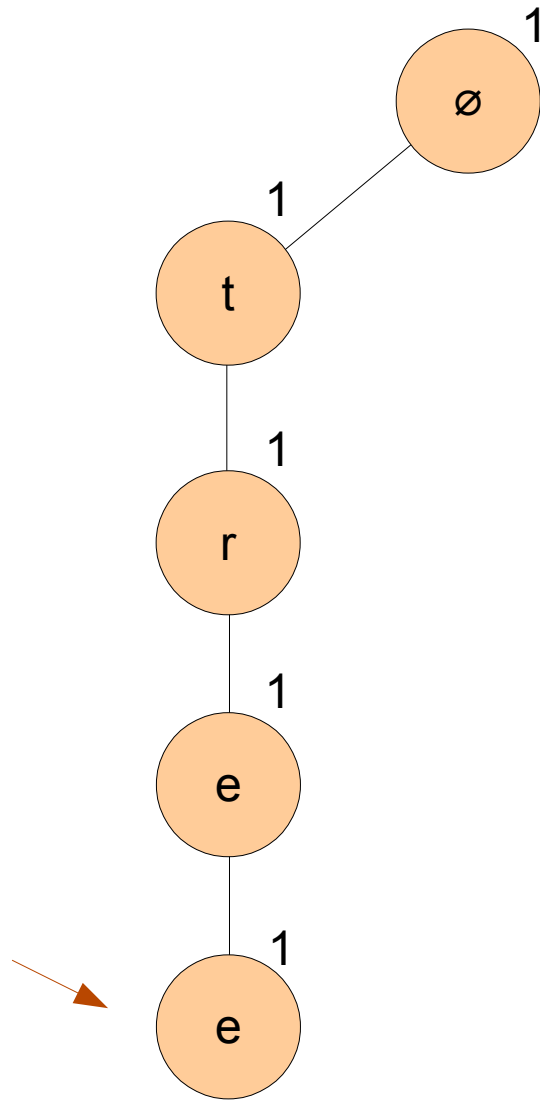
# Tries: Παράδειγμα Επέκτασης (2)

- Δυνατότητα απάντησης σε ερωτήματα “πόσες λέξεις αρχίζουν από ....;”
- Δημιουργία συνάρτησης `prefixCount` για αυτό τον σκοπό.

## Υλοποίηση:

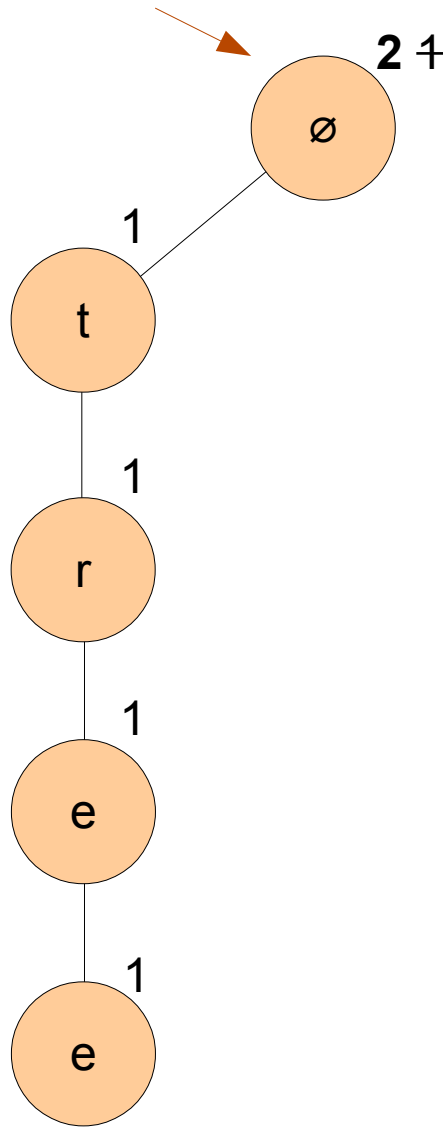
- Σε κάθε κόμβο του δέντρου θα αποθηκεύουμε έναν επιπλέον αριθμό: το πλήθος των λέξεων στις οποίες “ανήκει”.
- Θα πρέπει να συντηρούμε αυτά τα αθροίσματα σε κάθε πράξη `add` και `delete`.

# Tries: Παράδειγμα κατασκευής



→ add “tree”

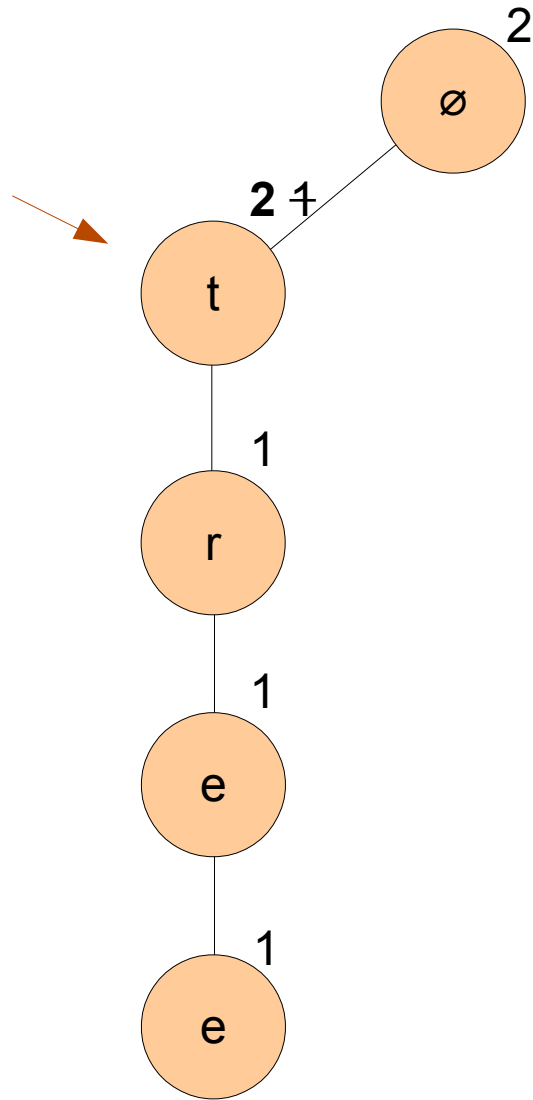
# Tries: Παράδειγμα κατασκευής



add "tree"  
→ add "trie"

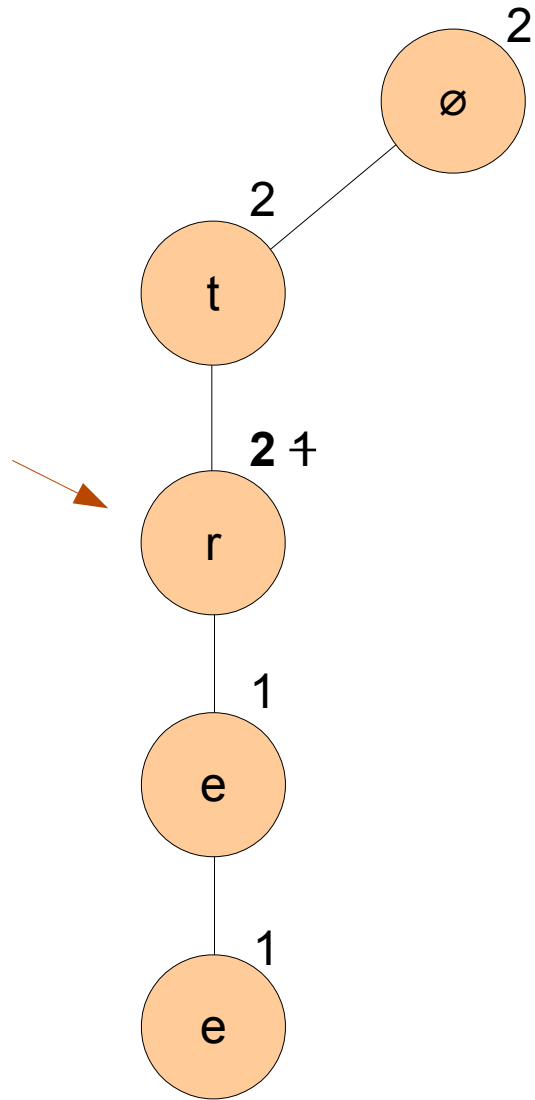


# Tries: Παράδειγμα κατασκευής



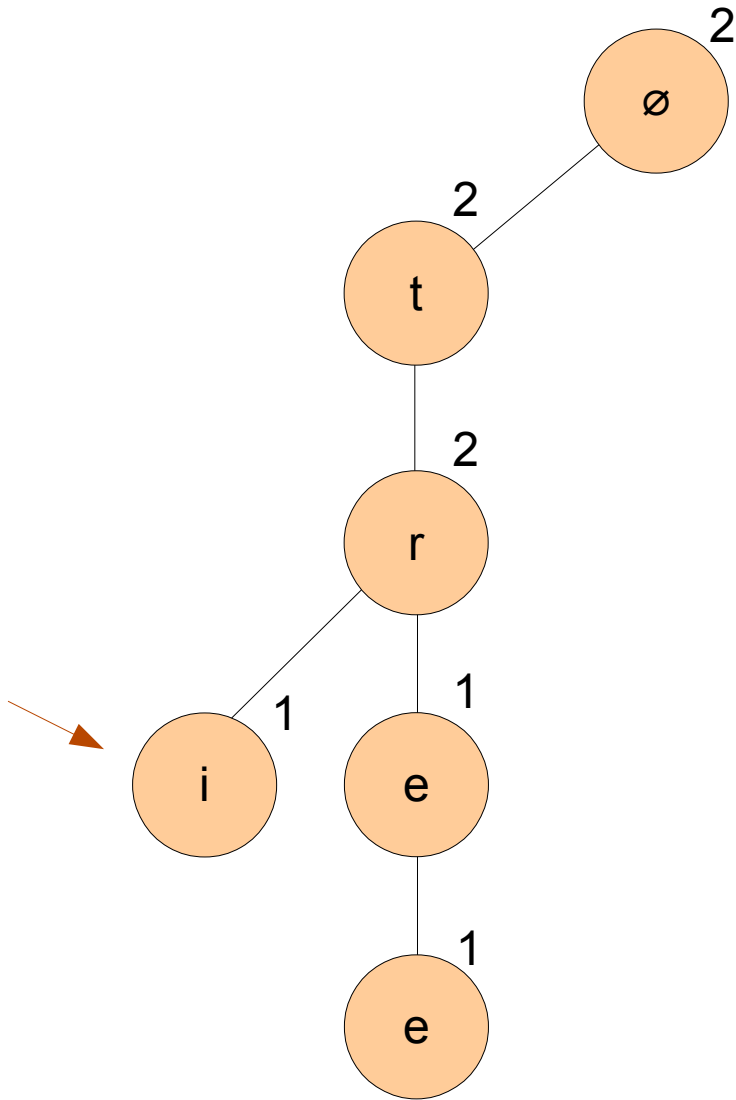
add "tree"  
add "trie"

# Tries: Παράδειγμα κατασκευής



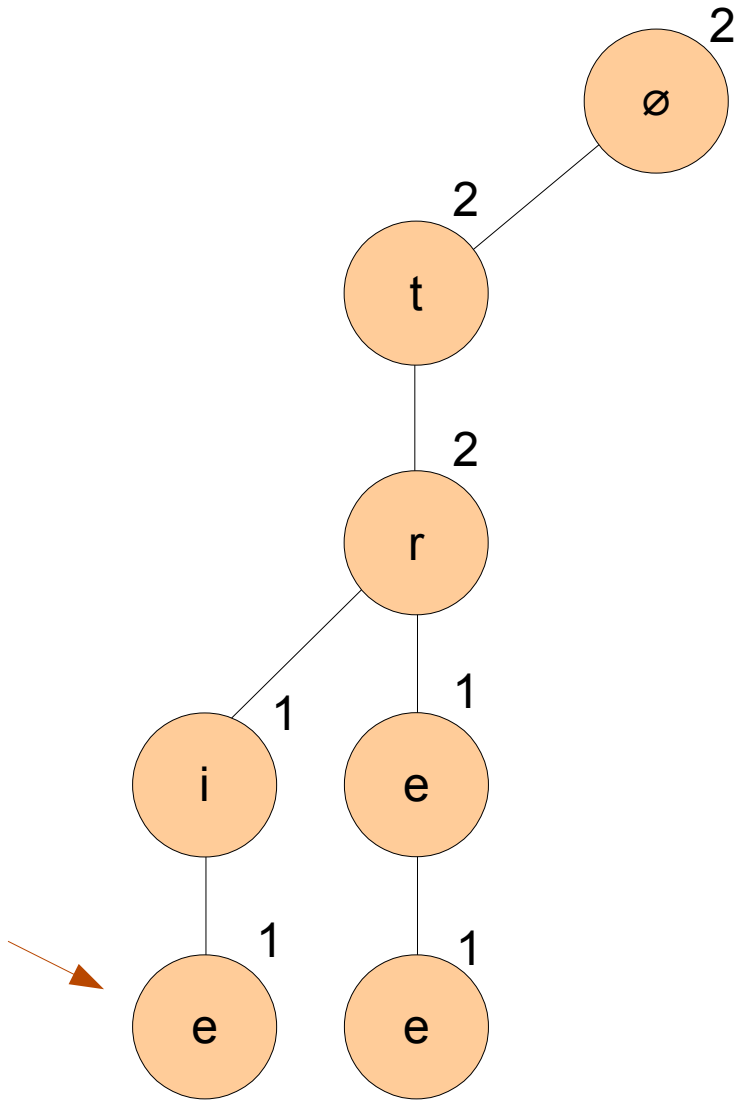
→ add "tree"  
add "trie"

# Tries: Παράδειγμα κατασκευής



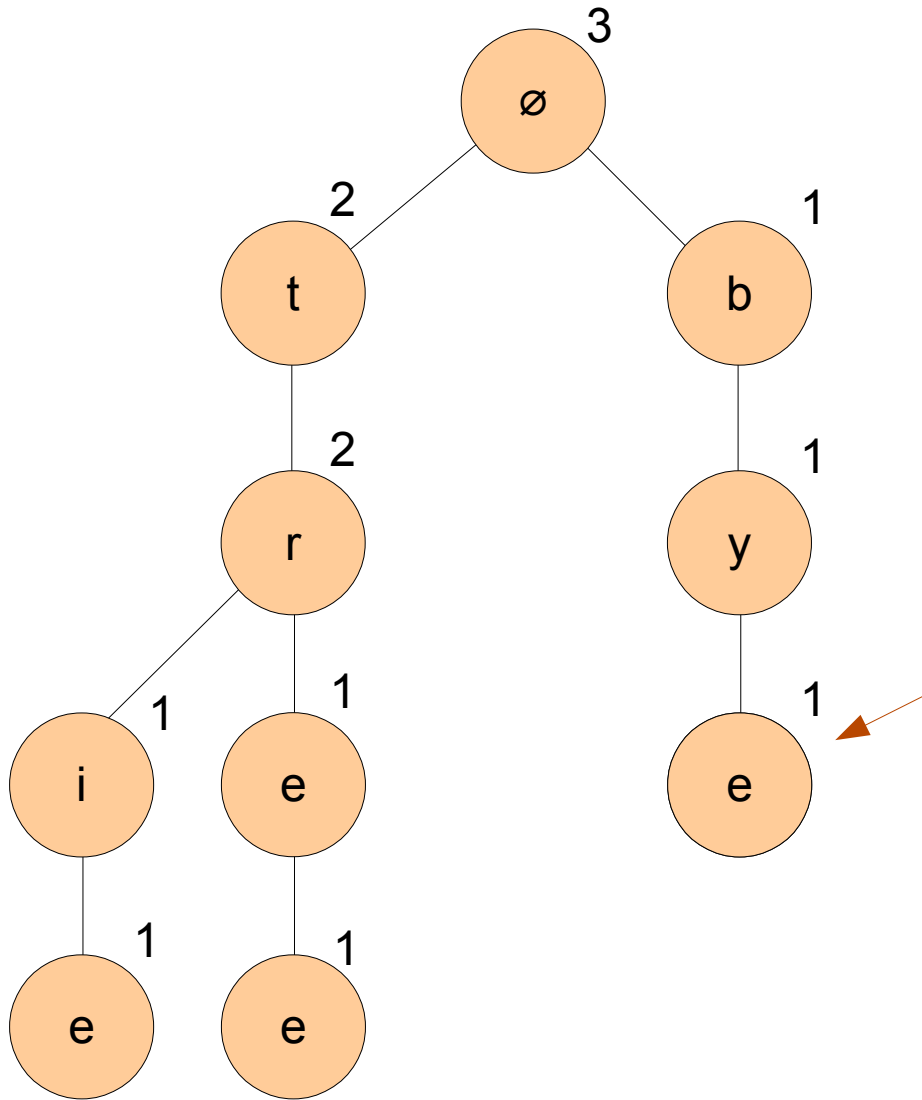
add "tree"  
→ add "trie"

# Tries: Παράδειγμα κατασκευής



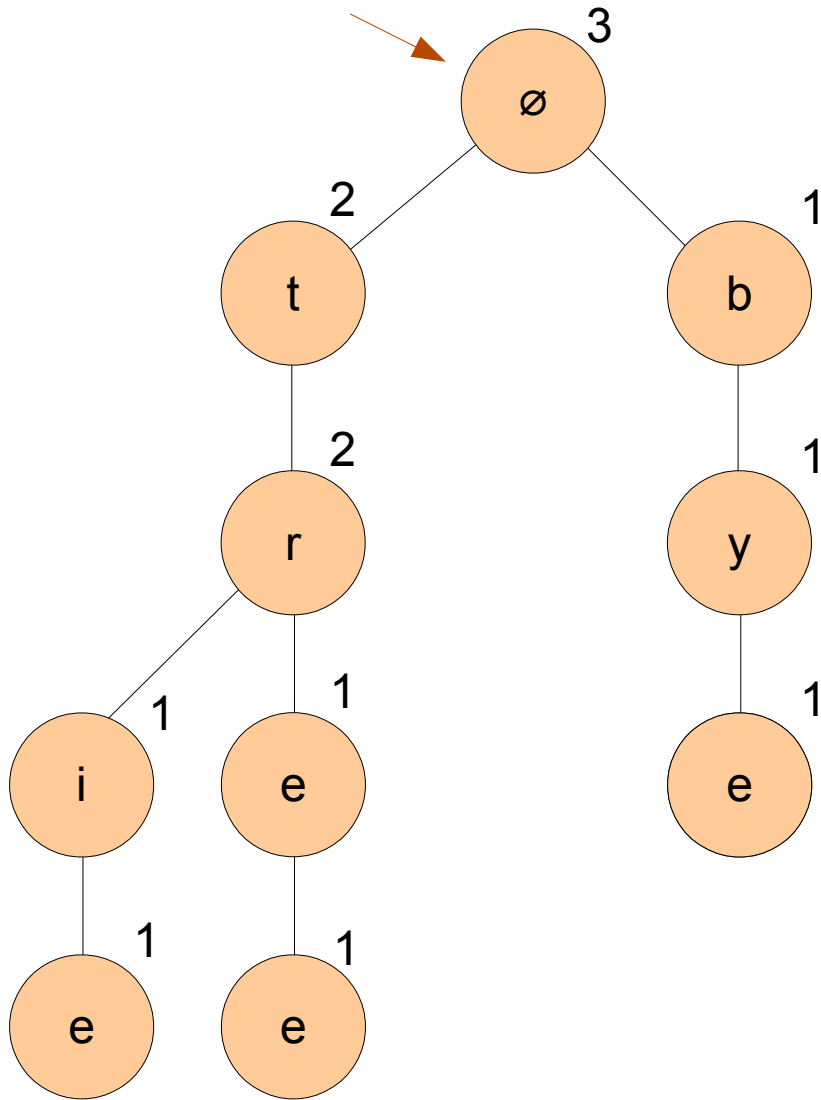
→ add "tree"  
→ add "trie"

# Tries: Παράδειγμα κατασκευής



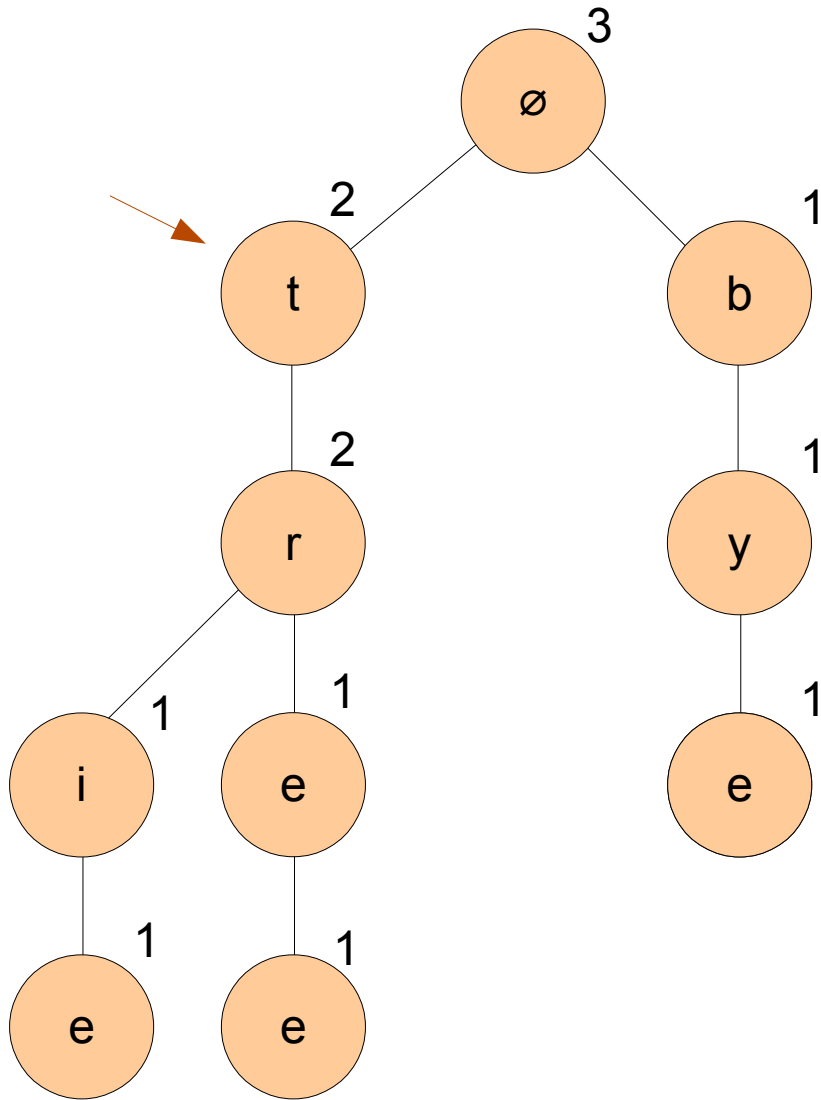
add "tree"  
add "trie"  
→ add "bye"

# Tries: Παράδειγμα κατασκευής



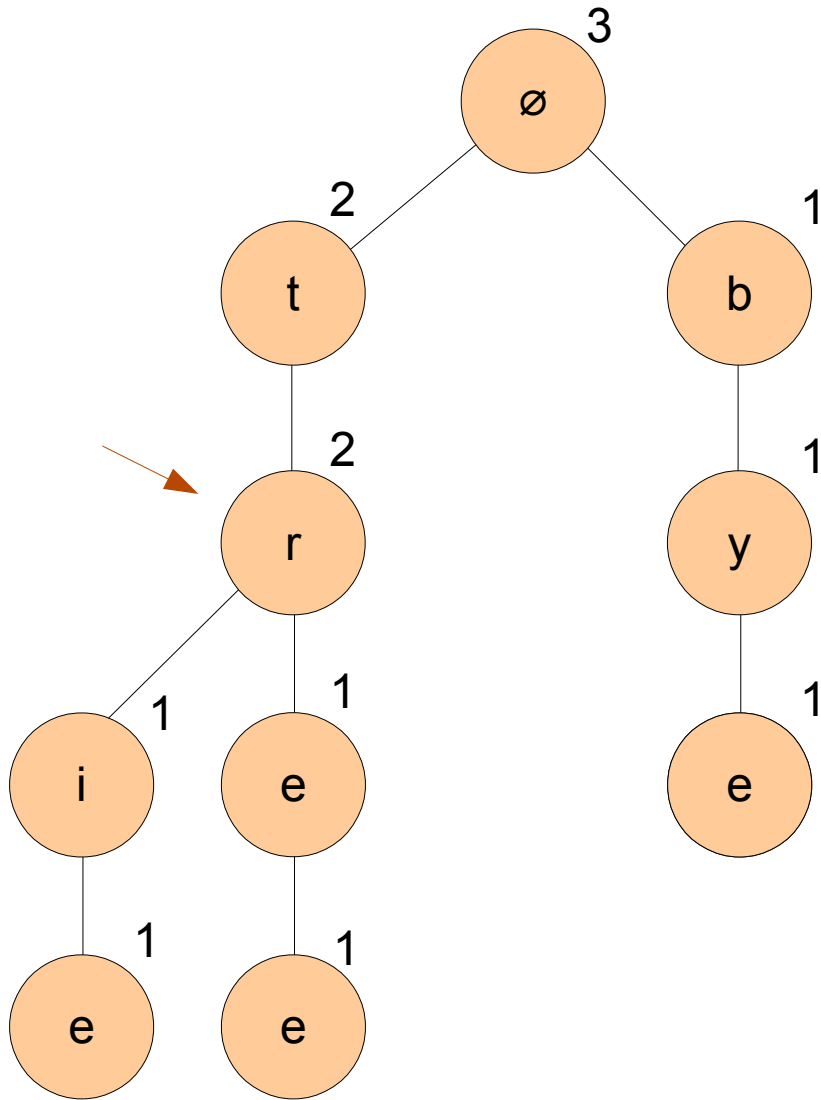
add "tree"  
add "trie"  
add "bye"  
→ pcount "tr"

# Tries: Παράδειγμα κατασκευής



add "tree"  
add "trie"  
add "bye"  
→ pcount "tr"

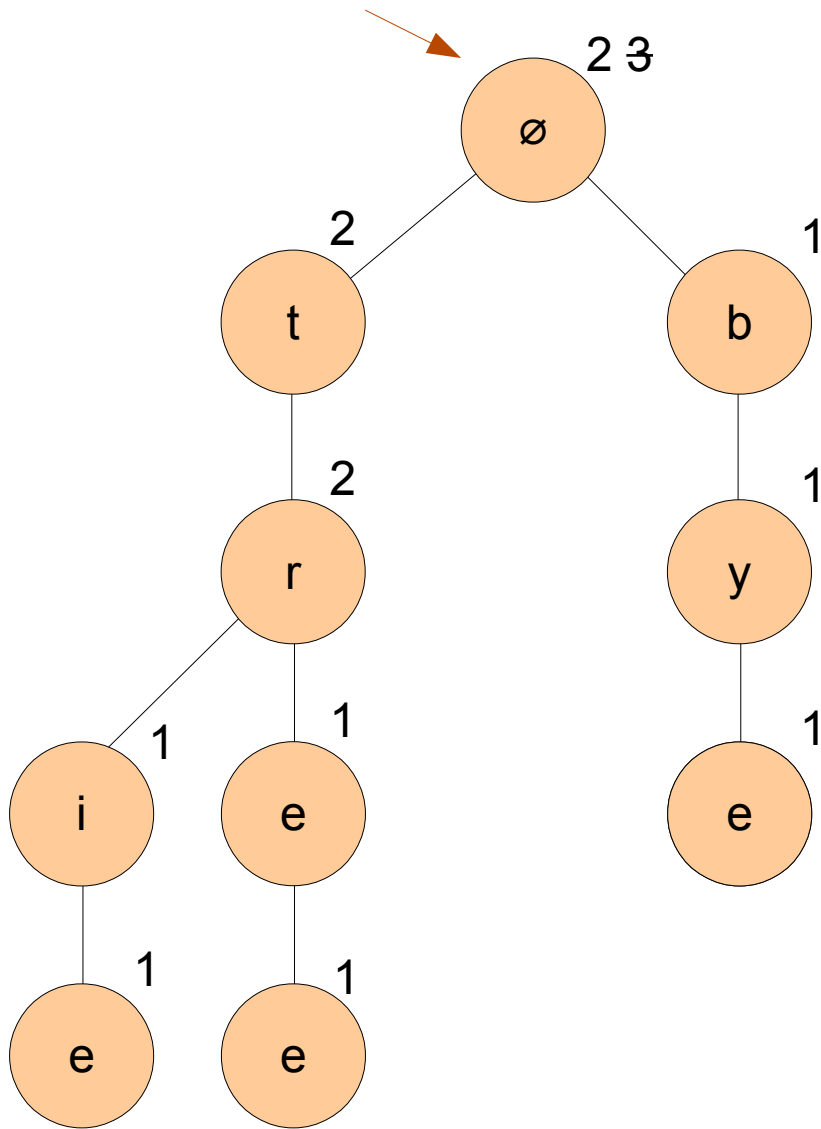
# Tries: Παράδειγμα κατασκευής



add "tree"  
add "trie"  
add "bye"  
→ pcount "tr" 2

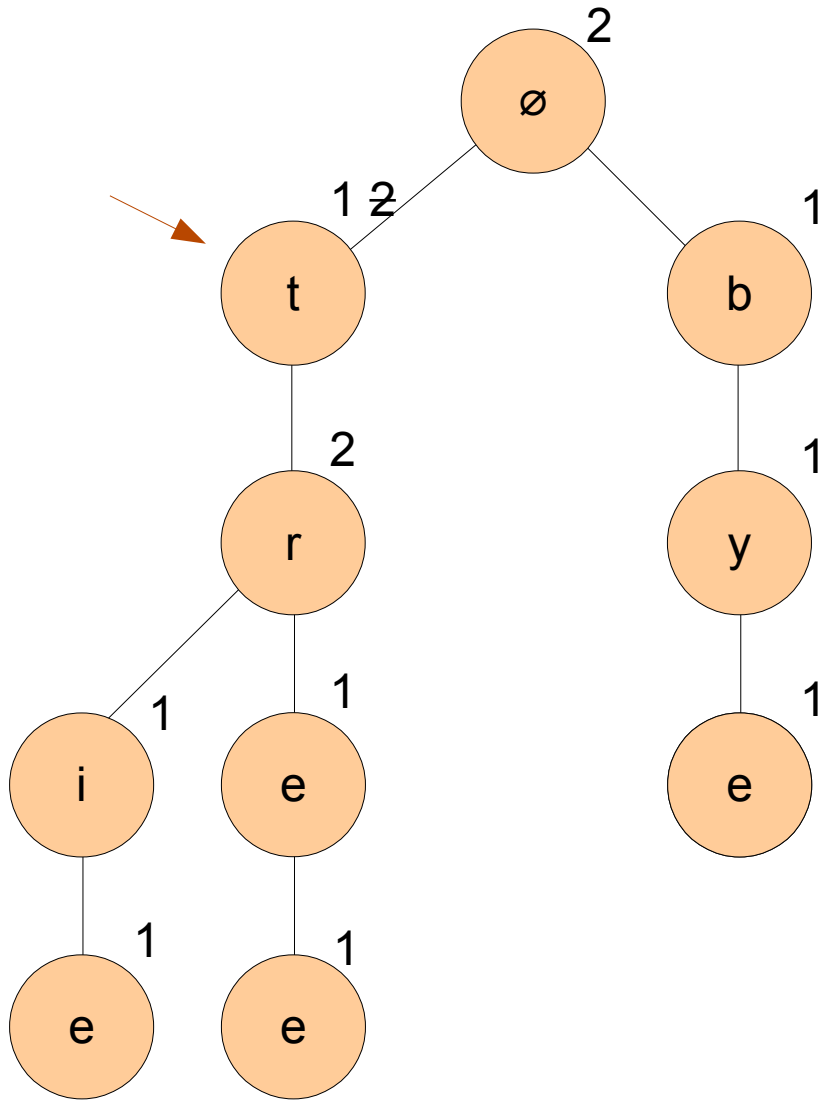


# Tries: Παράδειγμα κατασκευής



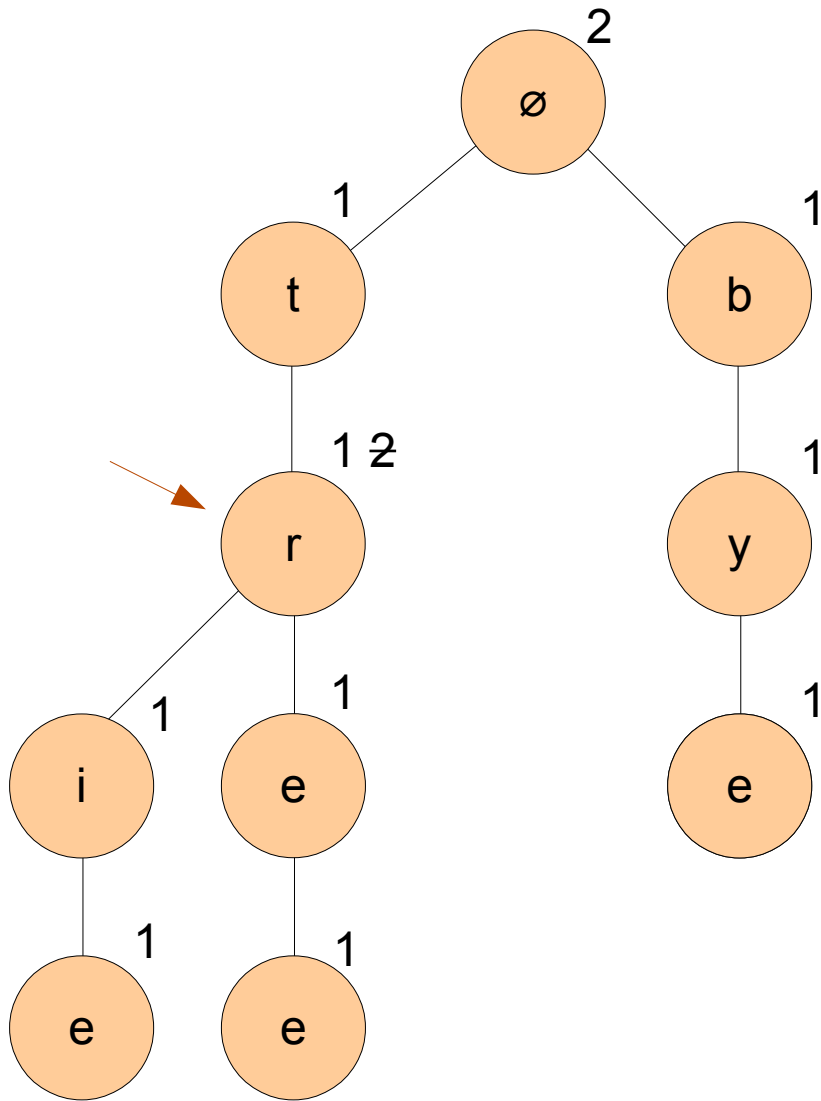
add "tree"  
add "trie"  
add "bye"  
pcount "tr" 2  
→ remove "trie"

# Tries: Παράδειγμα κατασκευής



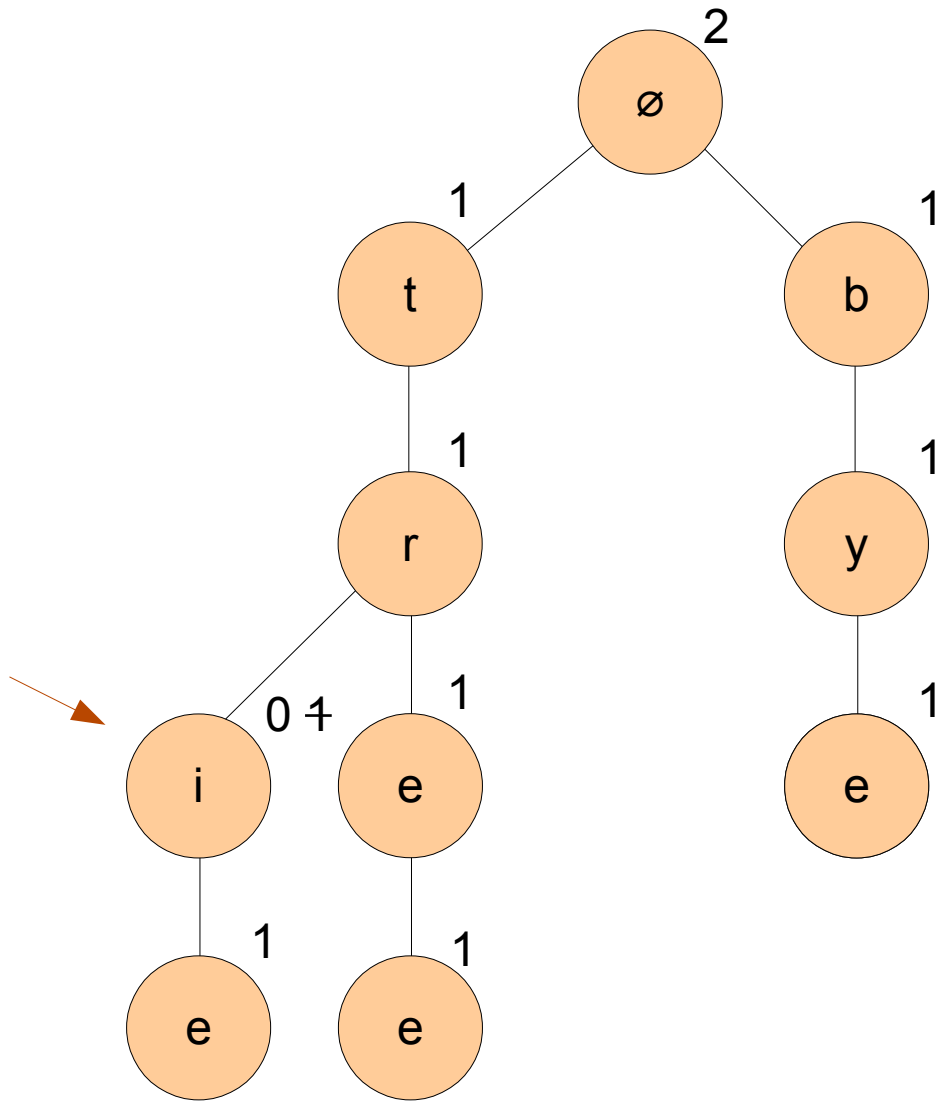
```
add "tree"  
add "trie"  
add "bye"  
pcount "tr" 2  
→ remove "trie"
```

# Tries: Παράδειγμα κατασκευής



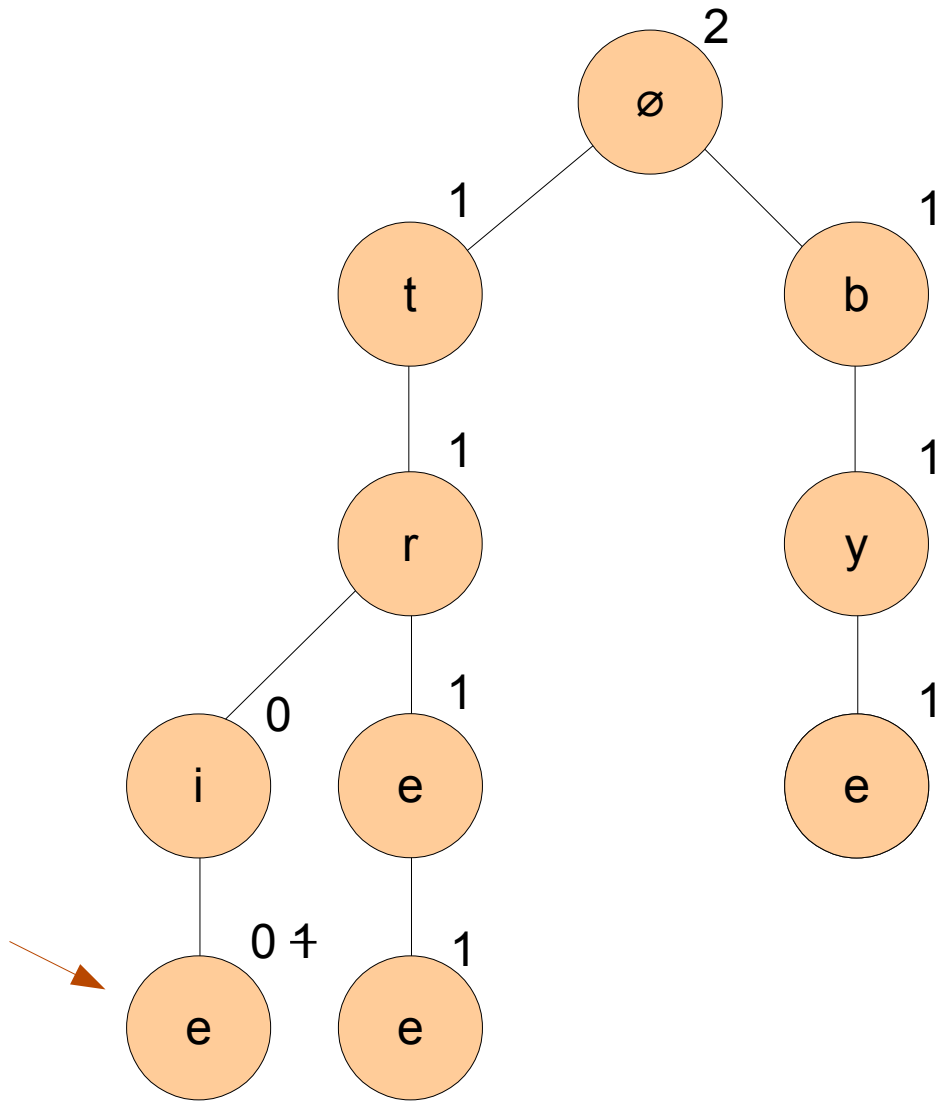
add "tree"  
add "trie"  
add "bye"  
pcount "tr" 2  
→ remove "trie"

# Tries: Παράδειγμα κατασκευής



add "tree"  
add "trie"  
add "bye"  
pcount "tr" 2  
→ remove "trie"

# Tries: Παράδειγμα κατασκευής



add "tree"  
add "trie"  
add "bye"  
pcount "tr" 2  
→ remove "trie"

# Tries: Υλοποίηση για μέτρηση προθεμάτων

```
#define N 50000
```

```
struct node {
```

```
    int children[26];
```

```
    int prefixes;
```

```
};
```

```
struct node Trie[N];
```


```
int trieNodeCount;
```

```
int initialize() {
```

```
    trieNodeCount = 1; /* προσθέτουμε την “ψεύτικη” ρίζα */
```

```
}
```


Μεταβλητή που αποθηκεύει το  
πλήθος των προθεμάτων που  
αντιστοιχούν σε αυτόν τον κόμβο



# Tries: Υλοποίηση της add

```
int add(char *word, int length) {  
    int i, nextNode, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα */  
  
    for (i=0; i<length; i++) {  
        nextNode = Trie[curNode].children[ word[i] - 'a' ];  
  
        if ( nextNode == 0 ) { /* αν δεν υπάρχει ο κόμβος στο  
                                trie, τότε τον δημιουργούμε */  
            trieNodeCount++;  
            Trie[curNode].children[ word[i] - 'a' ] = trieNodeCount;  
            curNode = trieNodeCount;  
        }  
        else {  
            curNode = nextNode;  
        }  
  
        Trie[curNode].prefixes++;  
    }  
}
```

Αυξάνουμε το prefixes για κάθε  
κόμβο στο μονοπάτι.



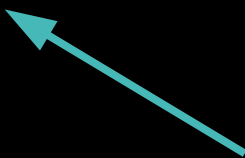
# Tries: Υλοποίηση της remove

```
int remove(char word, int length) { /* ακολουθούμε το μονοπάτι μέχρι τον
                                   κόμβο του τελευταίου γράμματος και στη
                                   συνέχεια αλλάζουμε το isWord σε 0 */

    int i, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα του trie */
    for (i=0; i<length; i++) {
        curNode = Trie[curNode].children[ word[i] - 'a' ];

        assert(curNode != 0); /* υποθέτουμε ότι η λέξη που θέλουμε να
                               διαγράψουμε υπάρχει πάντα */

        Trie[curNode].prefixes--;
    }
}
```



Αντίστοιχα εδώ μειώνουμε το πλήθος των προθεμάτων για κάθε κόμβο του μονοπατιού.



# Tries: Υλοποίηση της prefixCount

```
int prefixCount(char word, int length) {  
    int i, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα του trie */  
    for (i=0; i<length; i++) {  
        curNode = Trie[curNode].children[ word[i] - 'a' ];  
  
        if ( curNode == 0 ) { /* αν δεν υπάρχει ο κόμβος στο trie τότε  
            σίγουρα δεν υπάρχουν λέξεις με το prefix που ψάχνουμε */  
            return 0;  
        }  
    }  
    return Trie[curNode].prefixes;  
}
```

# Tries: Παραλλαγές

- Αν δεν επαρκεί η μνήμη για να αποθηκεύσουμε έναν πίνακα children μήκους  $|\Sigma|$  σε κάθε κόμβο, τότε μπορούμε να έχουμε έναν δυναμικό πίνακα που να περιέχει μόνο τα παιδιά που πράγματι υπάρχουν.
- Μπορούμε να αποθηκεύουμε μεγαλύτερα κομμάτια συμβολοσειρών σε κάθε κόμβο (αντί για ένα μόνο γράμμα).

# Binary Indexed Trees

## Πρόβλημα:

Έχουμε  $N$  κουτιά αριθμημένα από το 1 ως το  $N$ .  
Θέλουμε να φτιάξουμε ένα πρόγραμμα που να υποστηρίζει τις πράξεις:

- **ADD S X**: Πρόσθεσε  $X$  σπέρτα στο κουτί  $S$ .
- **SUM X Y**: Βρες το άθροισμα των σπέρτων που βρίσκονται από το κουτί  $X$  έως το κουτί  $Y$ . ( $X \leq Y$ )

# Binary Indexed Trees

## Προφανής Λύση:

Έχουμε έναν πίνακα  $N$  ακεραίων. Στην  $i$ -οστη θέση του πίνακα είναι αποθηκευμένο το πλήθος των σπίρτων που βρίσκονται στο  $i$ -οστο κουτί.

- **ADD S X**: Απλά προσθέτουμε τον αριθμό  $X$  στην θέση  $S$  του πίνακα. Αυτό φυσικά απαιτεί σταθερό χρόνο.
- **SUM X Y**: Πρέπει να προσθέσουμε όλες τις θέσεις από την  $X$  έως και την  $Y$ . Στην χειρότερη περίπτωση θα πρέπει να εκτελέσουμε  $N$  προσθέσεις. Επομένως η πολυπλοκότητα της SUM είναι  $O(N)$ .

# Binary Indexed Trees

Λύση με μερικά αθροίσματα:

Έχουμε έναν πίνακα  $\Pi$  από  $N$  ακεραίους. Στην  $i$ -οστή θέση του πίνακα είναι αποθηκευμένο το άθροισμα των σπάρτων που βρίσκονται από το 1ο μέχρι και το  $i$ -οστό κουτί.

- **ADD S X**: Θα πρέπει να προσθέσουμε το  $X$  σε όλες τις θέσεις από την  $S$  έως και την  $N$ .

# Binary Indexed Trees

## Λύση με μερικά αθροίσματα:

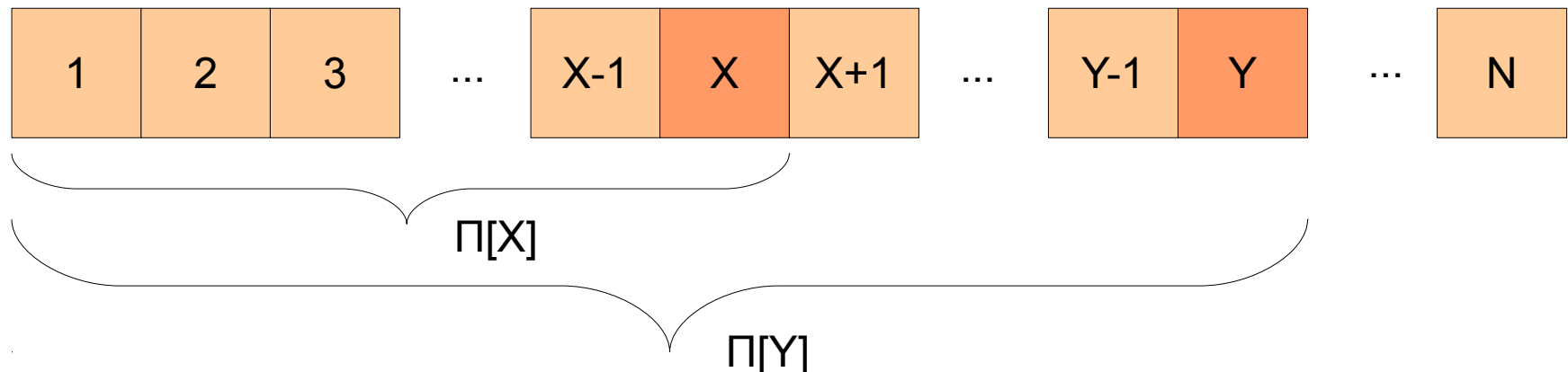
- **SUM X Y**: Το στοιχείο της θέσης Y (δηλαδή το  $\Pi[Y]$ ) περιέχει το άθροισμα:

$\text{κουτί}[1] + \text{κουτί}[2] + \text{κουτί}[3] + \text{κουτί}[X-1] + \text{κουτί}[X] + \dots + \text{κουτί}[Y]$

Από αυτό θέλουμε να αφαιρέσουμε τα:

$\text{κουτί}[1] + \text{κουτί}[2] + \text{κουτί}[3] + \text{κουτί}[X-1]$

Που αντιστοιχούν στο στοιχείο  $\Pi[X-1]$



# Binary Indexed Trees

## Λύση με μερικά αθροίσματα:

- Πιο προχωρημένη ιδέα, αλλά:
- ADD:  $O(N)$
- SUM:  $O(1)$
- Δεν κερδίσαμε τίποτα

# Binary Indexed Trees

## Λύση με buckets:

- Χωρίζουμε τον πίνακα σε  $O(N)$  buckets.
- ADD:  $O(1)$
- SUM:  $O(\sqrt{N})$



# Binary Indexed Trees

Λύσεις:

	<b>ADD</b>	<b>SUM</b>
Προφανής Λύση	$O(1)$	$O(N)$
Μερικά Αθροίσματα	$O(N)$	$O(1)$
Buckets	$O(1)$	$O(\sqrt{N})$
<b>Binary Indexed Trees</b>	<b><math>O(\log N)</math></b>	<b><math>O(\log N)</math></b>

# Binary Indexed Trees

- Είναι μια δομή δεδομένων που μας επιτρέπει να διαχειριζόμαστε μερικά αθροίσματα. Συγκεκριμένα υποστηρίζει τις πράξεις:
  - Πρόσθεση ενός αριθμού σε μια συγκεκριμένη θέση.
  - Άθροισμα των αριθμών που βρίσκονται σε ένα συγκεκριμένο διάστημα.

# Binary Indexed Trees

- Ουσιαστικά είναι ένας πίνακας αριθμών. Σε κάθε θέση του πίνακα αποθηκεύεται το άθροισμα των σπάρτων που βρίσκονται στο κουτί αυτής της θέσης καθώς και σε κάποιες από τις αμέσως προηγούμενες.

π.χ.

- Στην 12η θέση του πίνακα αποθηκεύονται τα σπάρτα που βρίσκονται από το 9ο έως και το 12ο κουτί.
- Αντίστοιχα στην 14η θέση του πίνακα αποθηκεύεται το πλήθος των σπάρτων που βρίσκονται από το 13ο έως και το 14ο κουτί.

# Binary Indexed Trees

- Πως αποφασίζουμε τα σπίρτα πόσων κουτιων θα αποθηκεύουμε σε κάθε θέση;
- Εξαρτάται αποκλειστικά από τον αριθμό της θέσης.
- Μετατροπή στο δυαδικό σύστημα.
- Κάθε θέση αποθηκεύει τόσα κουτιά όσα δείχνει ο μικρότερος άσσος στην δυαδική αναπαράσταση.

# Binary Indexed Trees

- $12_{10} = 1100_2$  . Αν κρατήσουμε μόνο τα ψηφία από τον τελευταίο άσσο και δεξιά, τότε έχουμε  $100_2 = 4_{10}$  . Άρα στην θέση 12 αποθηκεύουμε τα κουτιά των 4 τελευταίων θέσεων.

Δηλαδή από το 9ο έως το 12ο κουτί.

# Binary Indexed Trees

- $40_{10} = 101000_2$  . Αν κρατήσουμε μόνο τα ψηφία από τον τελευταίο άσσο και δεξιά, τότε έχουμε  $1000_2 = 8_{10}$  . Άρα στην θέση 40 αποθηκεύουμε τα κουτιά των 8 τελευταίων θέσεων.

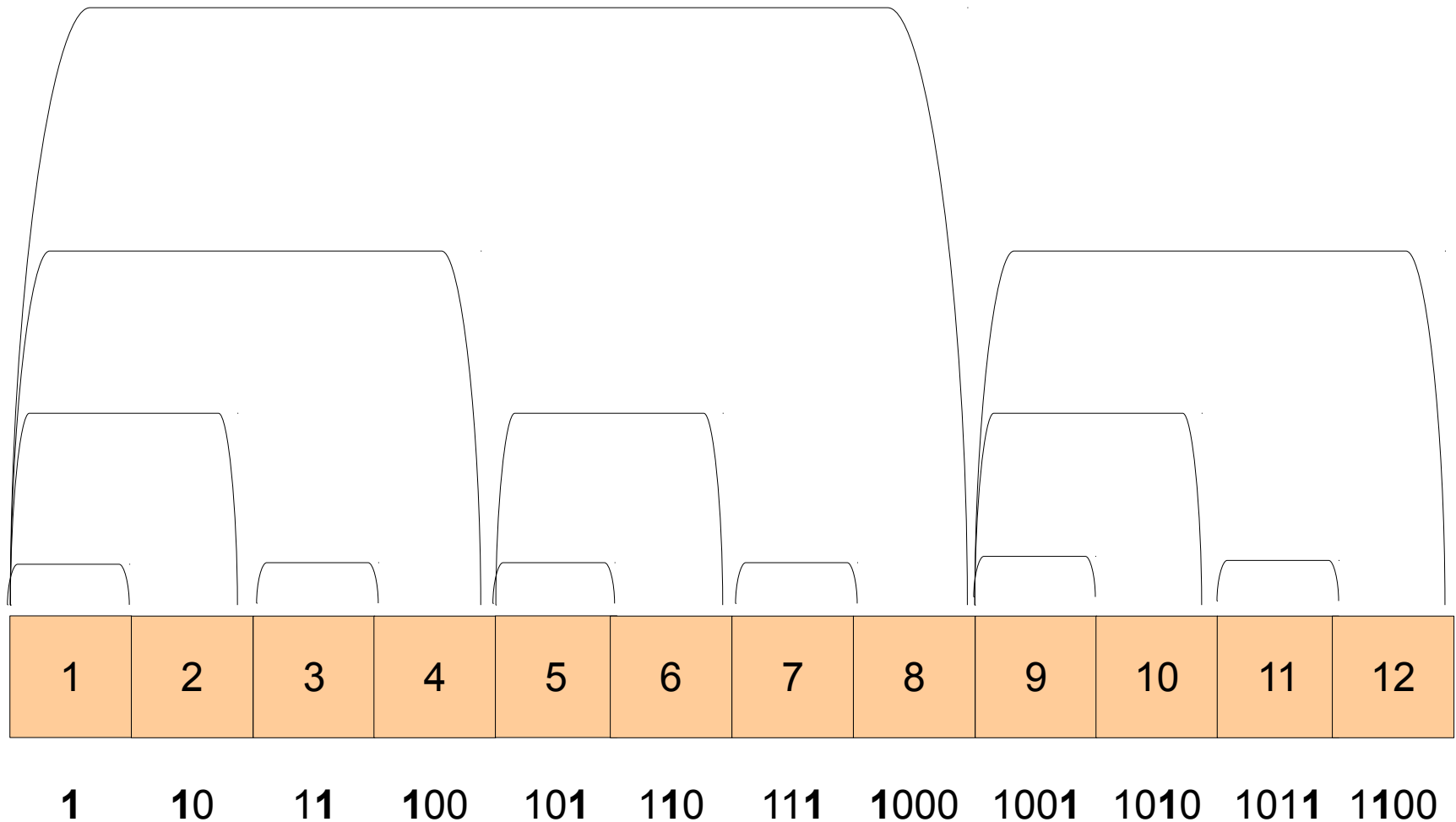
Δηλαδή από το 33ο έως το 40ο κουτί.

# Binary Indexed Trees

- $41_{10} = 101001_2$ . Αν κρατήσουμε μόνο τα ψηφία από τον τελευταίο άσσο και δεξιά, τότε έχουμε  $1_2 = 1_{10}$ . Άρα στην θέση 40 αποθηκεύουμε το κουτί μόνο της τελευταίας θέσης.

Δηλαδή μόνο το 41ο κουτί.

# Binary Indexed Trees





# Binary Indexed Trees

Θέση	Περιέχει
1	1
2	1 ... 2
3	3
4	1 ... 4
5	5
6	5 ... 6
7	7
8	1 ... 8
9	9
10	9 ... 10
11	11
12	9 ... 12
13	13
14	13 ... 14
15	15
16	1 ... 16

# Binary Indexed Trees

- Πώς απαντάμε στα ερωτήματα **SUM 1 Y**;
  - **Βήμα 1**  
Προσθέτουμε την τιμή της θέσης Y.
  - **Βήμα 2**  
Αφαιρούμε τον μικρότερο (δεξιότερο) άσσο από την δυαδική αναπαράσταση του Y. (π.χ.  $1010 \rightarrow 1000$ )
  - **Βήμα 3**  
Επαναλαμβάνουμε μέχρι το Y να γίνει 0.

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	0	0	0	4	6	3	2	8
BIT:	7	7	3	12	0	0	0	16	6	9	2	10

Έστω ότι θέλουμε να υπολογίσουμε το άθροισμα:  
 $\Sigma[1] + \Sigma[2] + \Sigma[3] + \dots + \Sigma[11]$

Θέλουμε, δηλαδή, να απαντήσουμε στο ερώτημα:  
**SUM 1 11**

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	0	0	0	4	6	3	2	8
BIT:	7	7	3	12	0	0	0	16	6	9	2	10



Αρχίζουμε από την θέση 11 και προσθέτουμε το  $\text{BIT}[11] = 2$  στο άθροισμα.

$\text{SUM} = 2$

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	0	0	0	4	6	3	2	8
BIT:	7	7	3	12	0	0	0	16	6	9	2	10



Η δυαδική αναπαράσταση του 11 είναι 1011. Αφαιρούμε τον πρώτο άσσο, ενώ τα υπόλοιπα ψηφία παραμένουν ίδια

$$\begin{array}{r}
 1011 \\
 - 0001 \\
 \hline
 1010 = 10_{10}
 \end{array}$$

SUM = 2

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	0	0	0	4	6	3	2	8
BIT:	7	7	3	12	0	0	0	16	6	9	2	10



Πλέον βρισκόμαστε στην θέση 10, οπότε προσθέτουμε την τιμή του BIT[10] στο άθροισμα.

$$\text{SUM} = 2 + 9 = 11$$

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	0	0	0	4	6	3	2	8
BIT:	7	7	3	12	0	0	0	16	6	9	2	10



Η δυαδική αναπαράσταση του 10 είναι 1010. Αφαιρούμε τον πρώτο άσσο, ενώ τα υπόλοιπα ψηφία παραμένουν ίδια

$$\begin{array}{r}
 1010 \\
 - 0010 \\
 \hline
 1000 = 8_{10}
 \end{array}$$

SUM = 11

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	0	0	0	4	6	3	2	8
BIT:	7	7	3	12	0	0	0	16	6	9	2	10



Τώρα είμαστε στην θέση 8, οπότε προσθέτουμε την τιμή του BIT[8] στο άθροισμα.

$$\text{SUM} = 11 + 16 = 27$$



# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	0	0	0	4	6	3	2	8
BIT:	7	7	3	12	0	0	0	16	6	9	2	10



Η δυαδική αναπαράσταση του 8 είναι 1000. Αφαιρούμε τον πρώτο άσσο, ενώ τα υπόλοιπα ψηφία παραμένουν ίδια

$$\begin{array}{r}
 1000 \\
 - 1000 \\
 \hline
 0000 = 0_{10}
 \end{array}$$

Άρα η τελική απάντηση είναι **SUM = 27**.

# Binary Indexed Trees

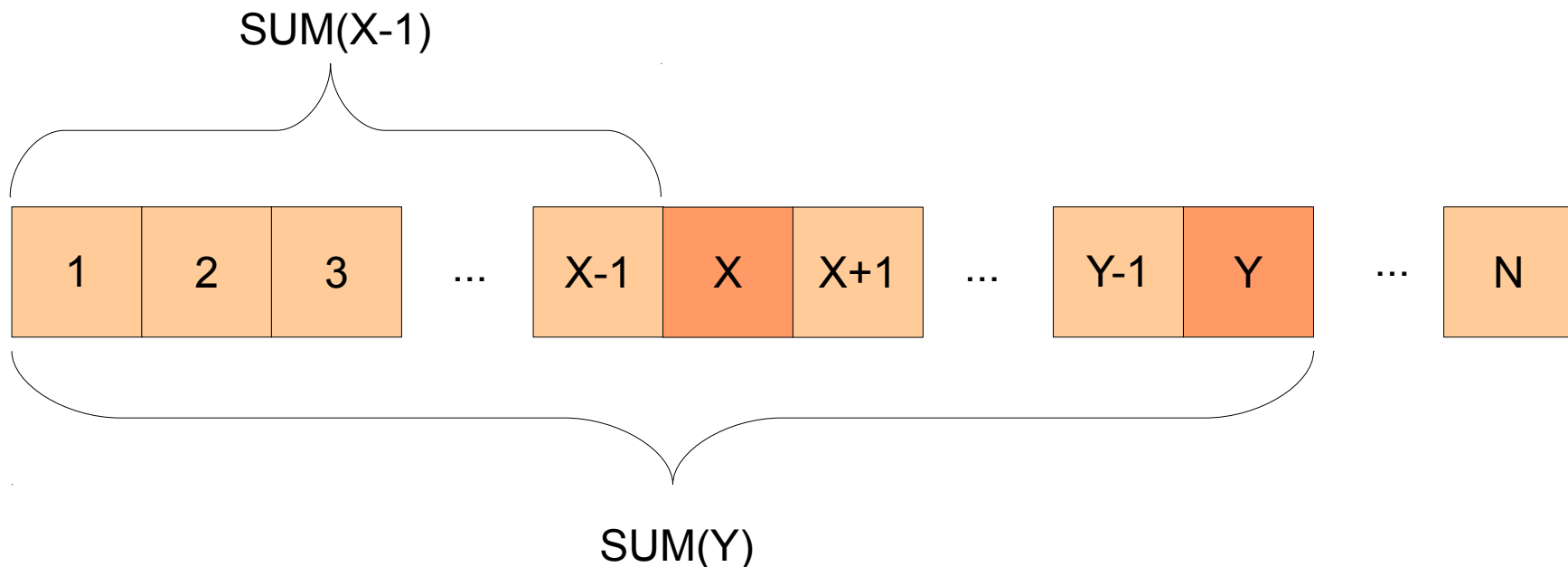
- **Ερώτηση:** Γιατί είναι  $O(\log N)$ ;
  - Ξεκινώντας από το  $Y$ , θα κάνουμε τόσες προσθέσεις όσοι είναι οι άσσοι του  $Y$  (αφού σε κάθε βήμα θα κάνουμε μια πρόσθεση και θα σβήνουμε έναν άσσο από την δυαδική αναπαράστασή του).
  - Η δυαδική αναπαράσταση κάθε αριθμού  $N$  αποτελείται από  $\log_2 N + 1$  ψηφία.
  - Θα εκτελέσουν το πολύ  $\log_2 N + 1 = O(\log N)$  προσθέσεις.

# Binary Indexed Trees

- Πως απαντάμε στο ερώτημα **SUM X Y**;

Το σπάμε σε δύο:

$$\text{SUM } X \ Y = \text{SUM } 1 \ Y - \text{SUM } 1 \ X-1$$

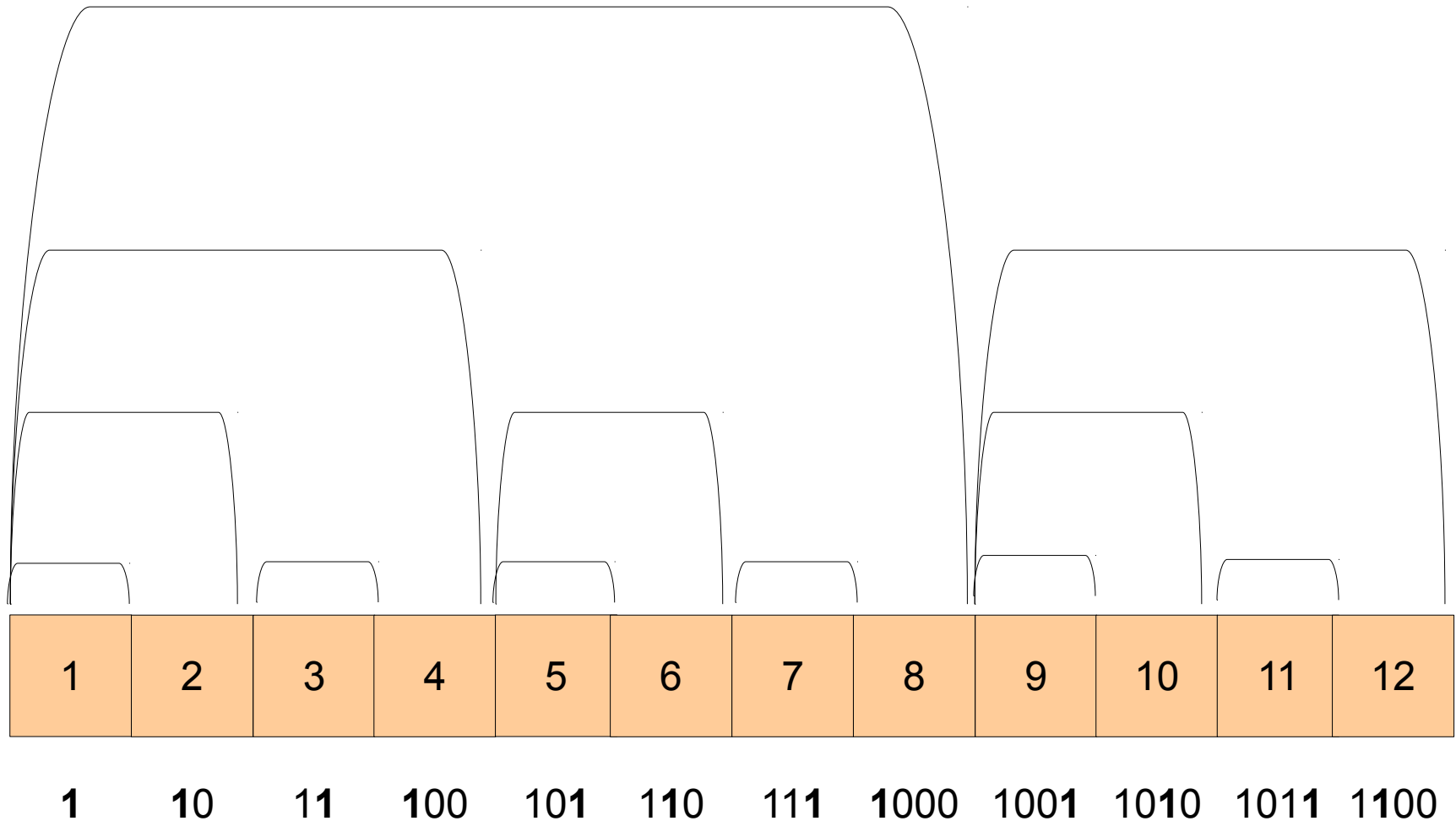


# Binary Indexed Trees

- Πώς απαντάμε στα ερωτήματα **ADD S X**;
  - Πρέπει να αλλάξουμε την τιμή όλων των θέσεων που περιέχουν την S.
  - Αντίστροφη Διαδικασία

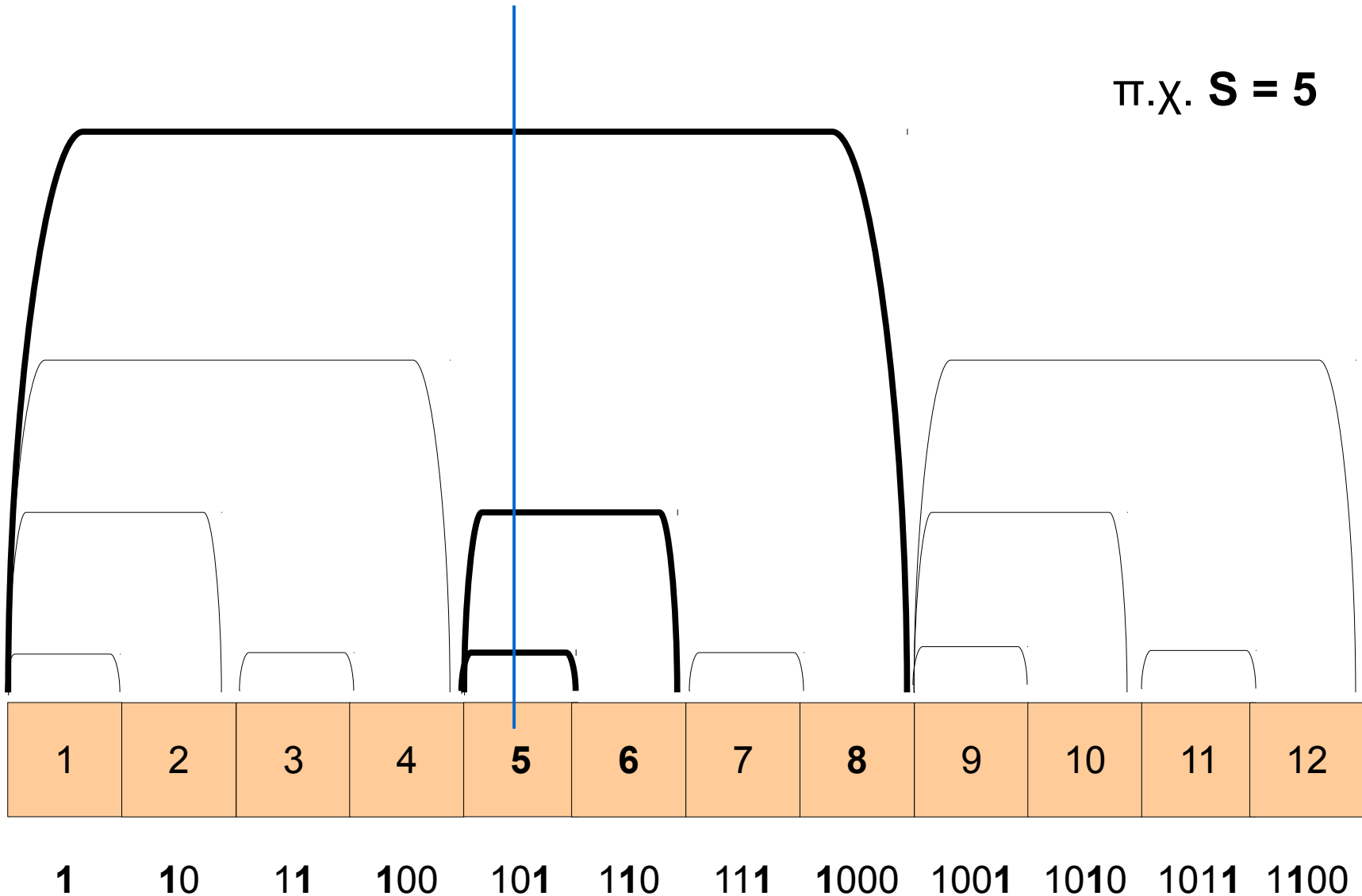
# Binary Indexed Trees

$\pi.x.S = 5$



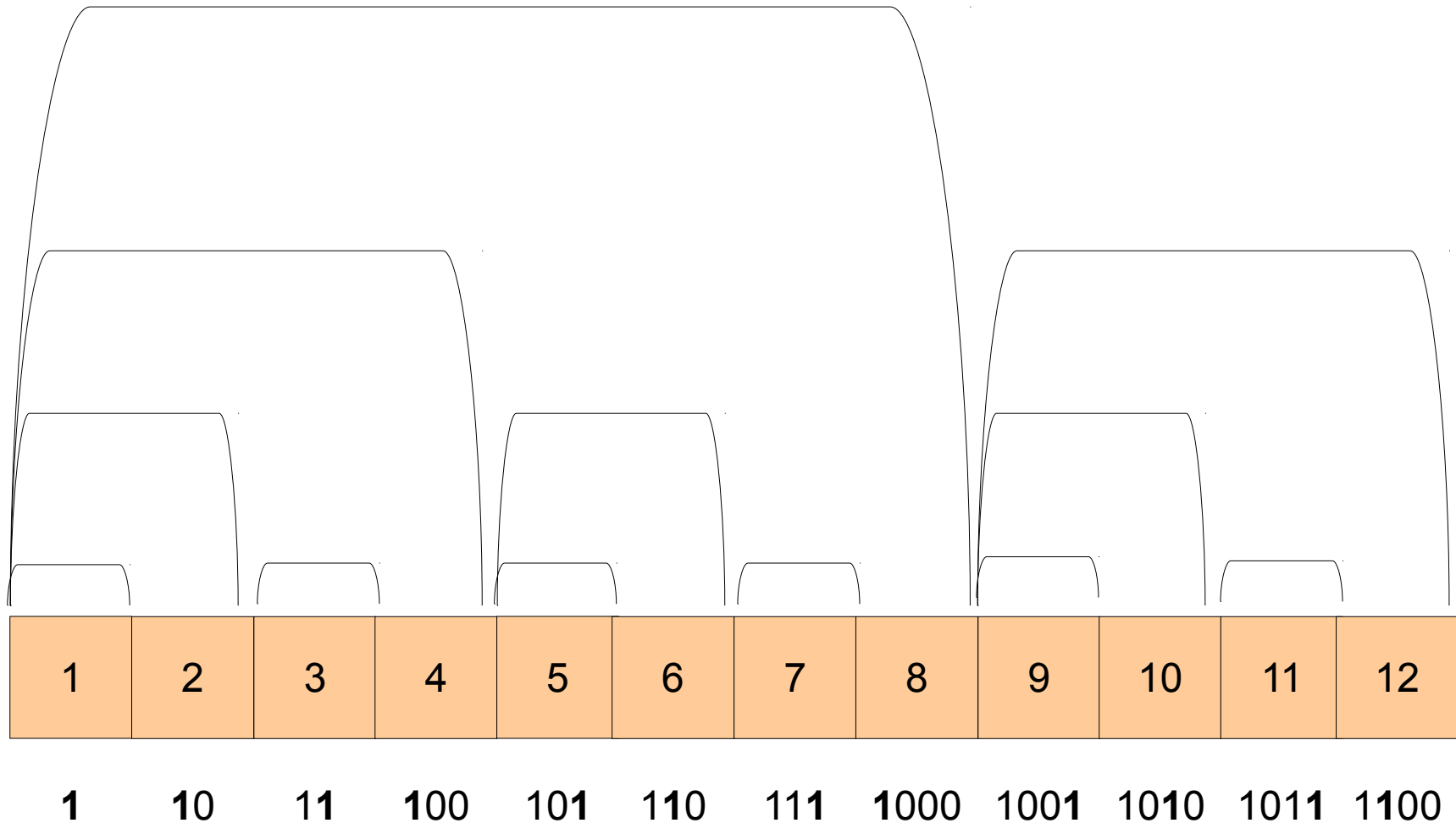
# Binary Indexed Trees

$\pi.X. S = 5$



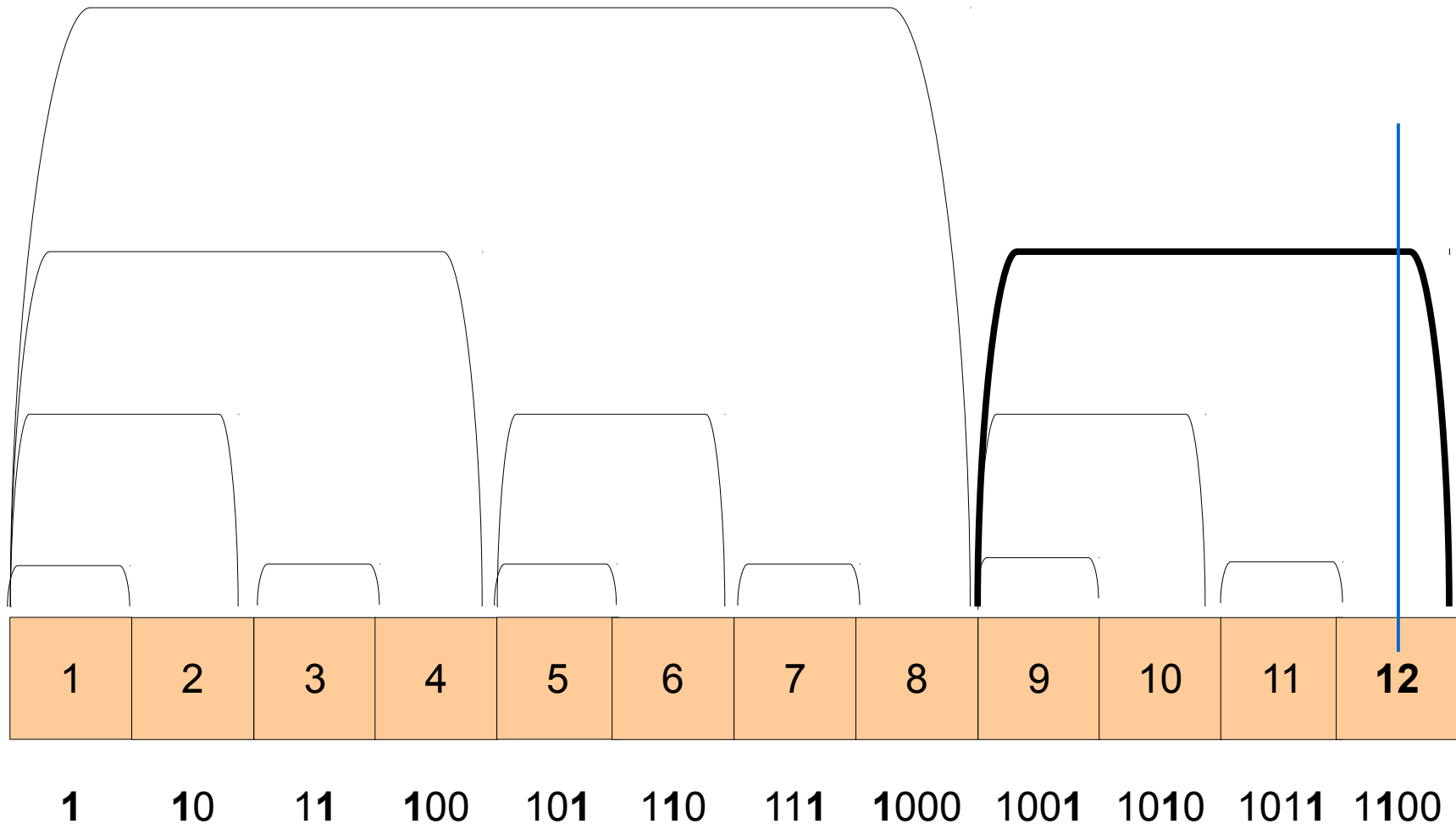
# Binary Indexed Trees

$\pi.X. S = 12$



# Binary Indexed Trees

$\pi.x$ . **S = 12**





# Binary Indexed Trees

- **Βήμα 1**

Προσθέτουμε τον αριθμό  $X$  στην θέση  $S$ .

- **Βήμα 2**

Προσθέτουμε τον μικρότερο (δεξιότερο) άσσο στην την δυαδική αναπαράσταση του  $S$ .

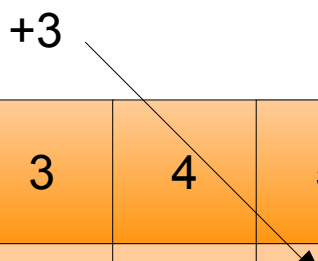
(π.χ.  $1010 \rightarrow 1100$ )

## **Βήμα 3**

Επαναλαμβάνουμε μέχρι το  $S$  να ξεπεράσει το μέγεθος του πίνακα.

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	0	0	0	4	6	3	2	8
BIT:	7	7	3	12	0	0	0	16	6	9	2	10




Έστω ότι θέλουμε να προσθέσουμε 3 σπίρτα στην θέση 5:

Θέλουμε, δηλαδή, να εκτελέσουμε την πράξη:

**ADD 5 3**

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	3	0	0	4	6	3	2	8
BIT:	7	7	3	12	<b>3</b>	0	0	16	6	9	2	10



Αρχικά προσθέτουμε το 3 στην θέση 5 του πίνακα BIT.

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	3	0	0	4	6	3	2	8
BIT:	7	7	3	12	<b>3</b>	0	0	16	6	9	2	10



Η δυαδική αναπαράσταση του 5 είναι 101. Προσθέτουμε,  
Λοιπόν, πρώτο άσσο:

$$\begin{array}{r} 101 \\ + 001 \\ \hline 110 \end{array} = 6_{10}$$

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	3	0	0	4	6	3	2	8
BIT:	7	7	3	12	<b>3</b>	<b>3</b>	0	16	6	9	2	10



Άρα η επόμενη θέση του πίνακα BIT στην οποία πρέπει να προστεθεί το 3 είναι η θέση 6.

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	3	0	0	4	6	3	2	8
BIT:	7	7	3	12	3	3	0	16	6	9	2	10



Η δυαδική αναπαράσταση του 6 είναι 110. Προσθέτουμε, ξανά πρώτο άσσο:

$$\begin{array}{r} 110 \\ + 010 \\ \hline 1000 = 8_{10} \end{array}$$

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	3	0	0	4	6	3	2	8
BIT:	7	7	3	12	3	3	0	19	6	9	2	10



Έτσι το προσθέτουμε το 3 στο BIT[8] και γίνεται:

$$\text{BIT}[8] = 16 + 3 = 19$$

# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	3	0	0	4	6	3	2	8
BIT:	7	7	3	12	3	3	0	<b>19</b>	6	9	2	10



Η δυαδική αναπαράσταση του 8 είναι **1000**. Προσθέτοντας τον πρώτο (και μοναδικό) άσσο παίρνουμε:

$$\begin{array}{r} 1000 \\ + 1000 \\ \hline 10000 = 16_{10} \end{array}$$



# Binary Indexed Trees

Θέση:	1	2	3	4	5	6	7	8	9	10	11	12
(Σπίρτα) Σ:	7	0	3	2	3	0	0	4	6	3	2	8
BIT:	7	7	3	12	3	3	0	<b>19</b>	6	9	2	10

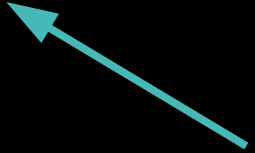
Το 16 όμως είναι μεγαλύτερο από το μέγεθος του πίνακα BIT, άρα η διαδικασία τερματίζει εδώ.

# Binary Indexed Trees: Υλοποίηση της sum

```
int sum(int Y) {  
    int answer = 0; /* μεταβλητή που κρατάει το ζητούμενο άθροισμα */  
    while (Y > 0) {  
        answer += BIT[Y];  
        Y -= (Y & -Y);  
    }  
    return answer;  
}
```

# Binary Indexed Trees: Υλοποίηση της sum

```
int sum(int Y) {  
    int answer = 0; /* μεταβλητή που κρατάει το ζητούμενο άθροισμα */  
    while (Y > 0) {  
        answer += BIT[Y];  
        Y -= (Y & -Y);  
    }  
    return answer;  
}
```

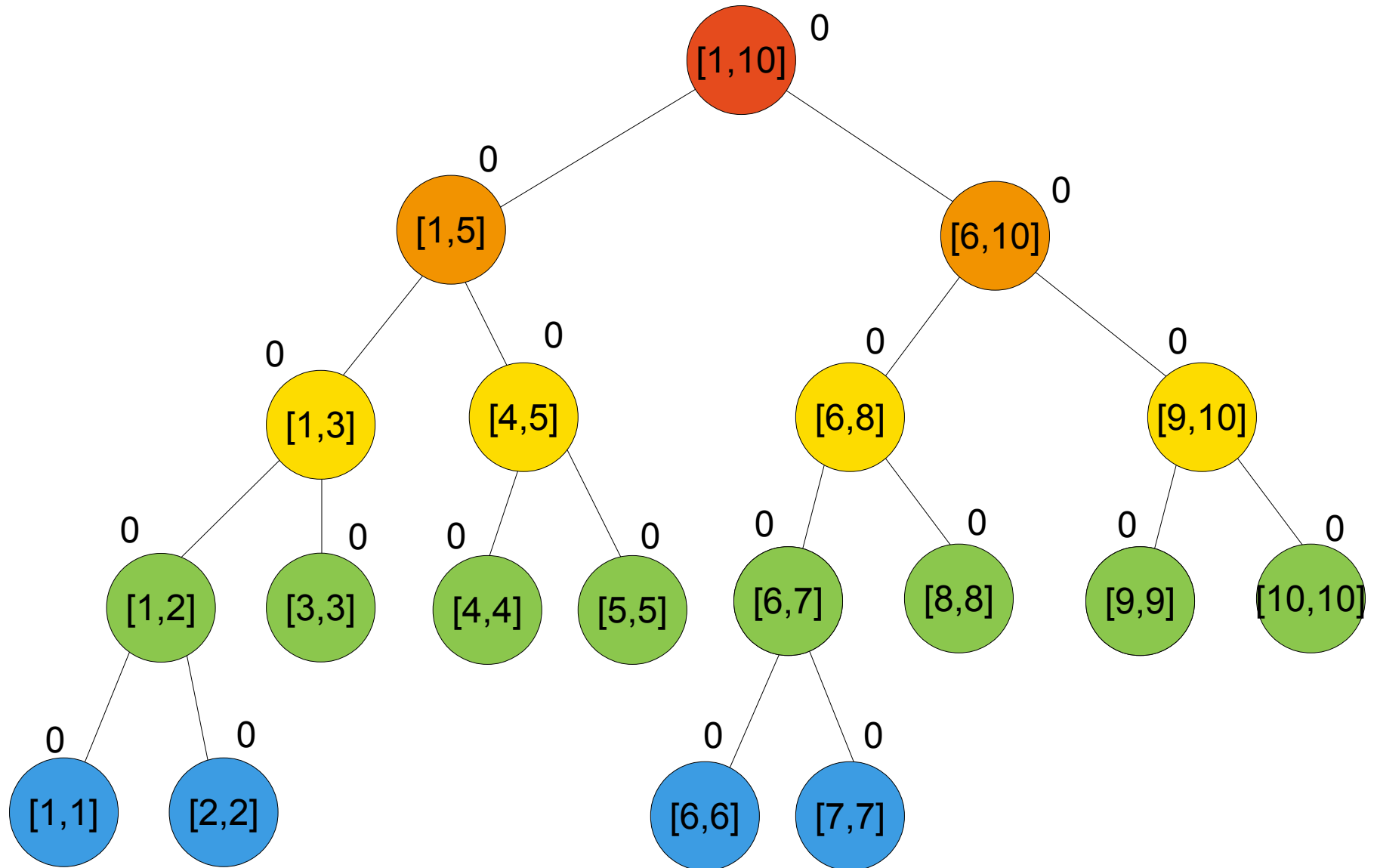


Τρικ

# Binary Indexed Trees: Υλοποίηση της add

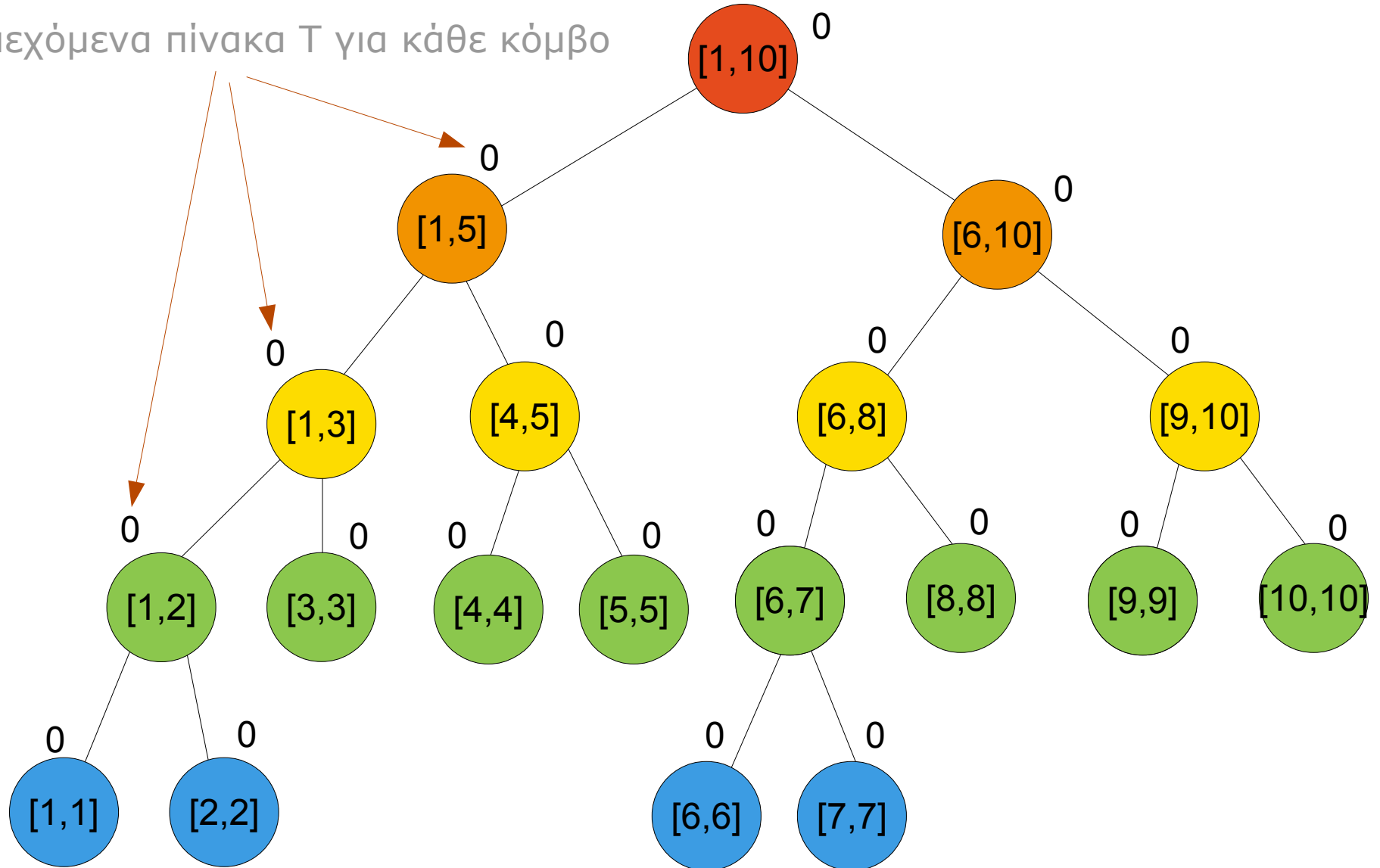
```
void add(int pos, int num) {  
    while (pos <= N) {  
        BIT[pos] += num;  
        pos += (pos & -pos);  
    }  
}
```

# Interval Trees



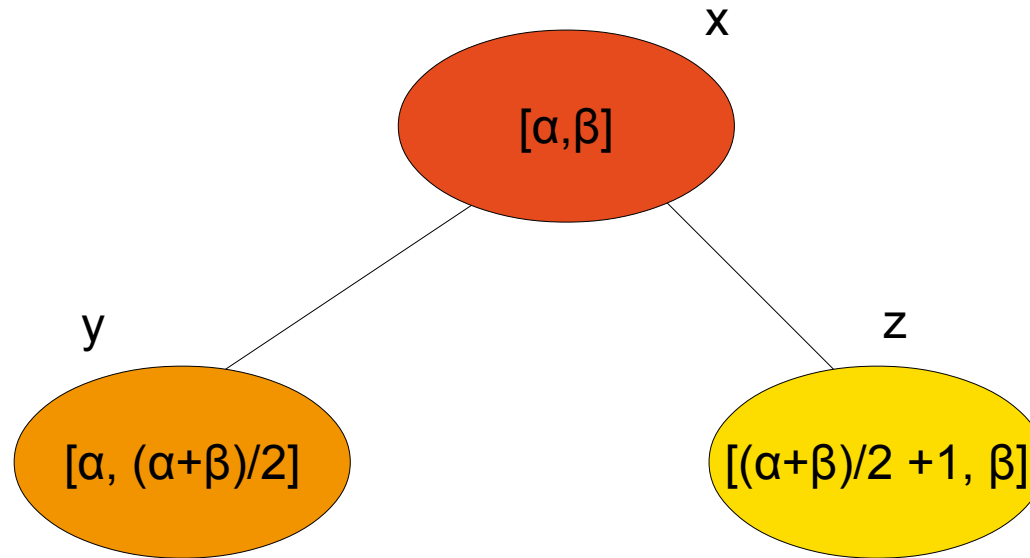
# Interval Trees

Περιεχόμενα πίνακα T για κάθε κόμβο

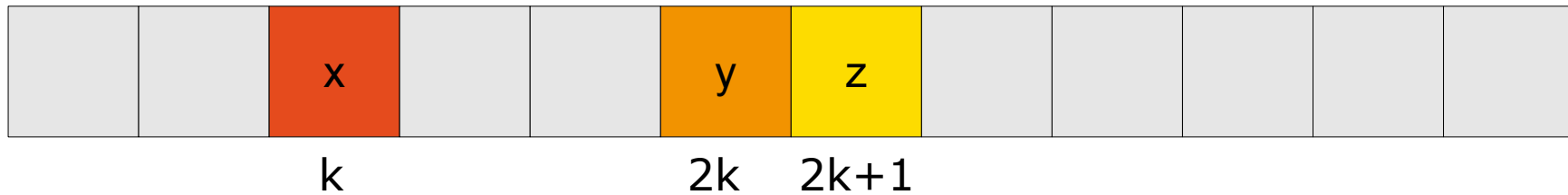


# Interval Trees

(αναπαράσταση στη μνήμη)



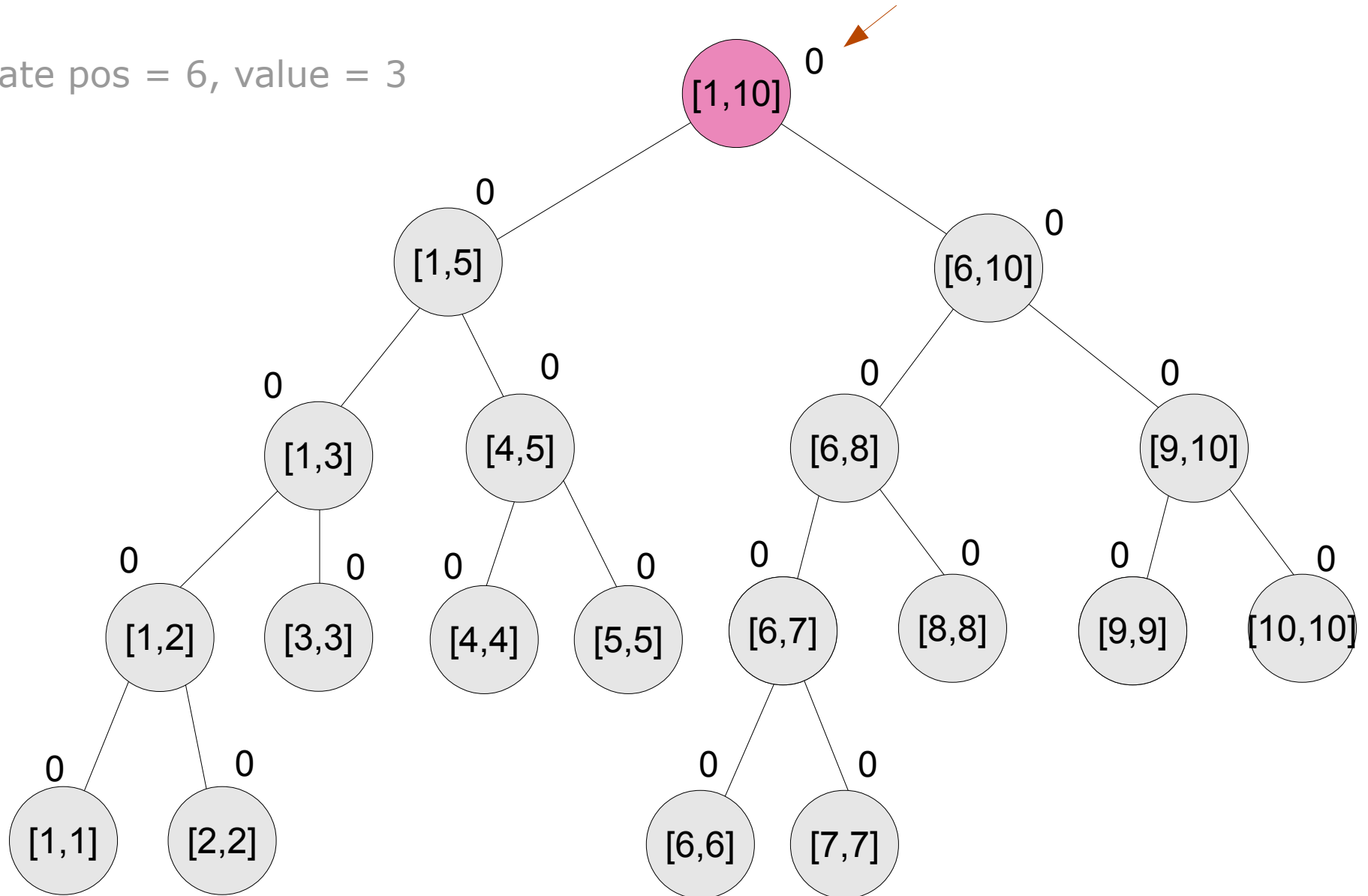
Πίνακας T:



Αριστερό παιδί του k :  $2 * k$   
Δεξιό παιδί του k:  $2 * k + 1$   
Πατέρας του k:  $k / 2$

# Interval Trees

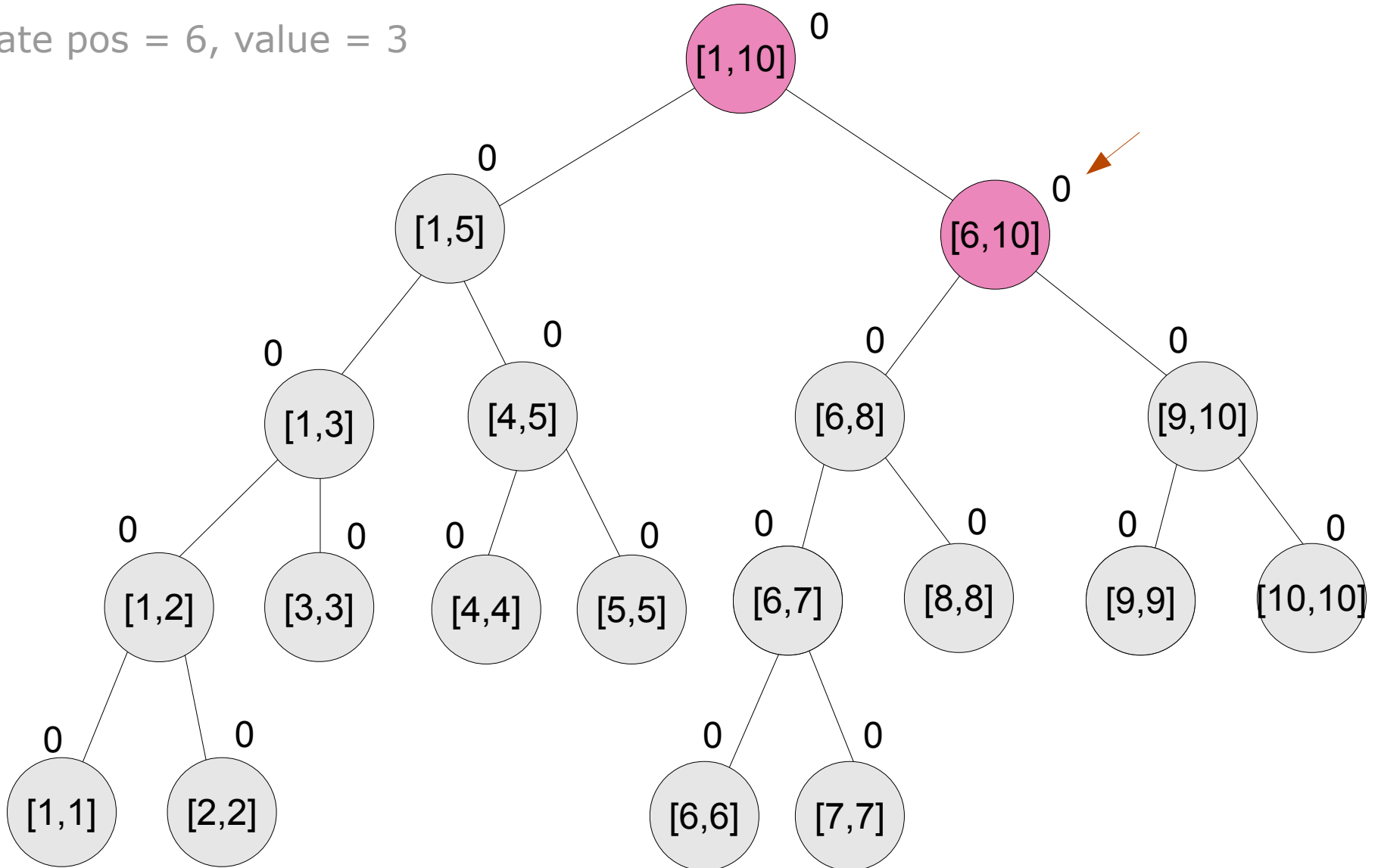
Update pos = 6, value = 3





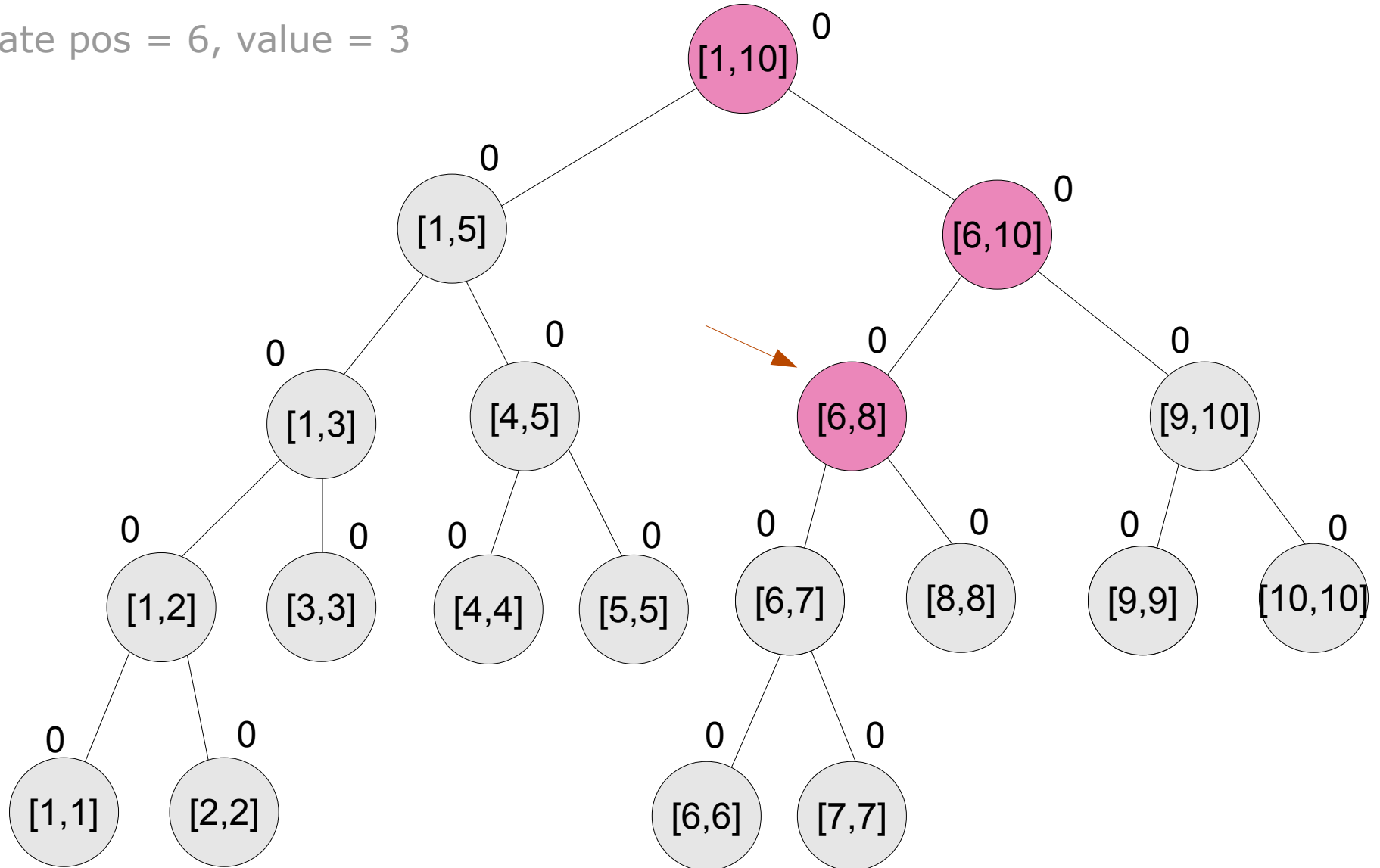
# Interval Trees

Update pos = 6, value = 3



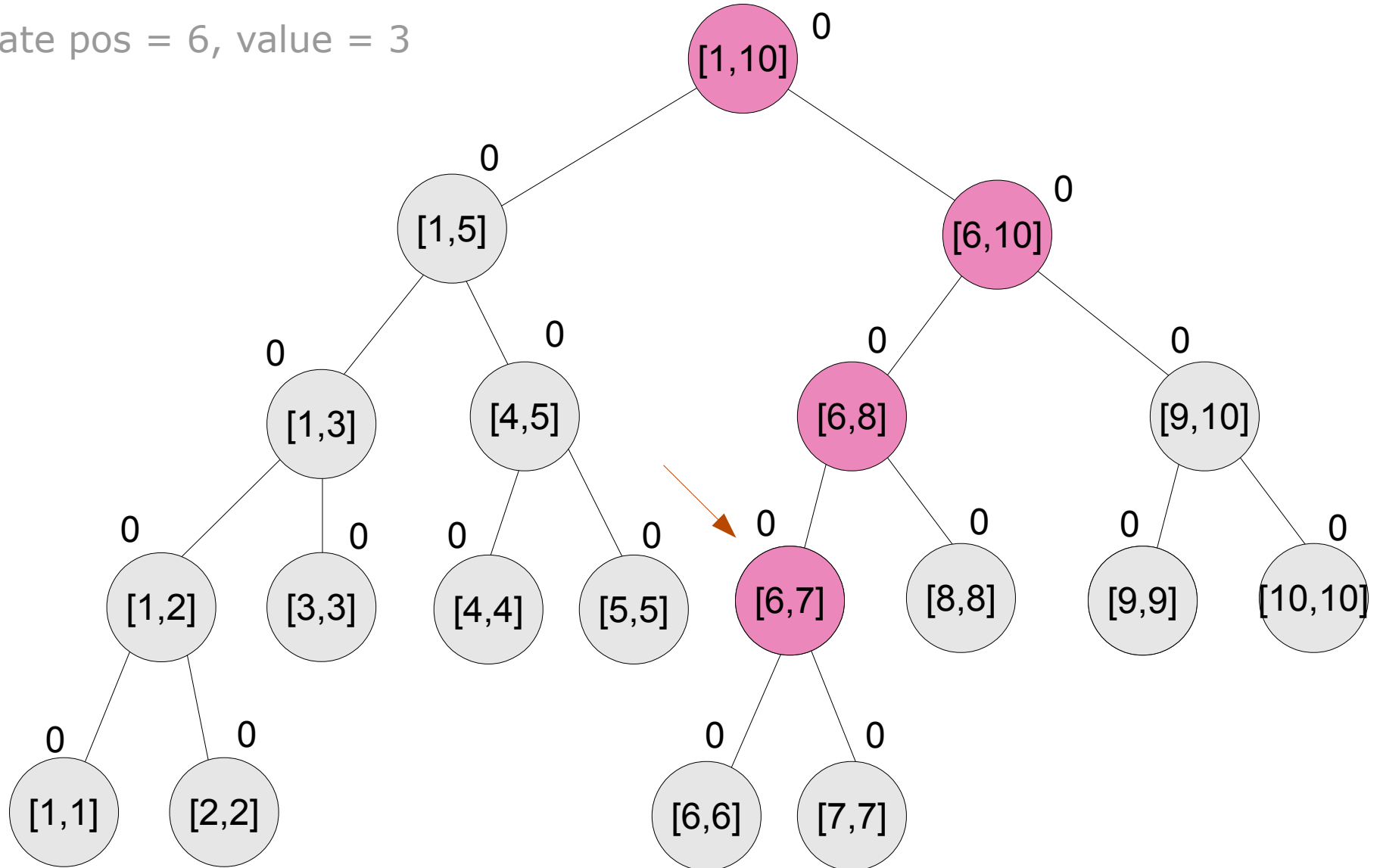
# Interval Trees

Update pos = 6, value = 3



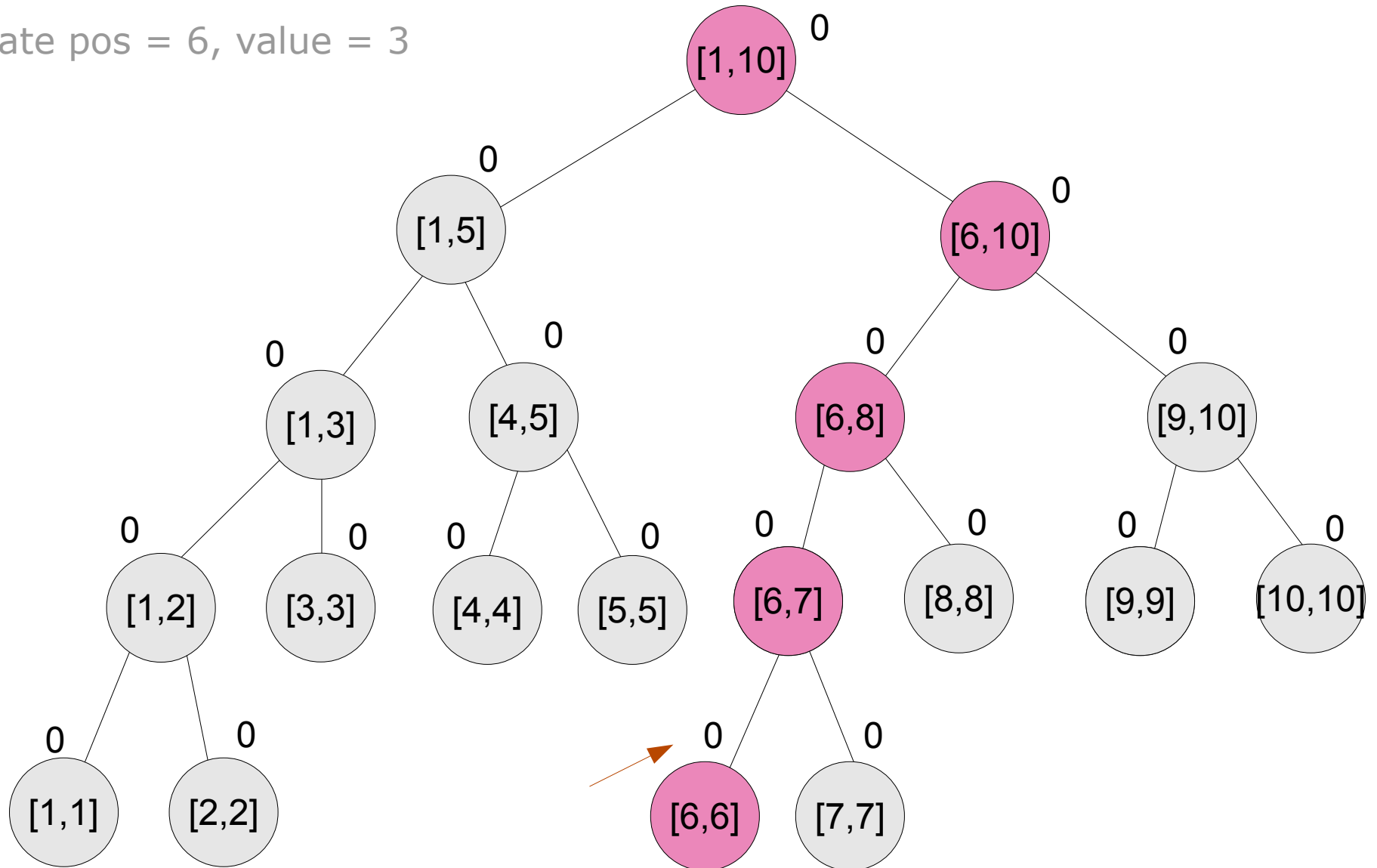
# Interval Trees

Update pos = 6, value = 3



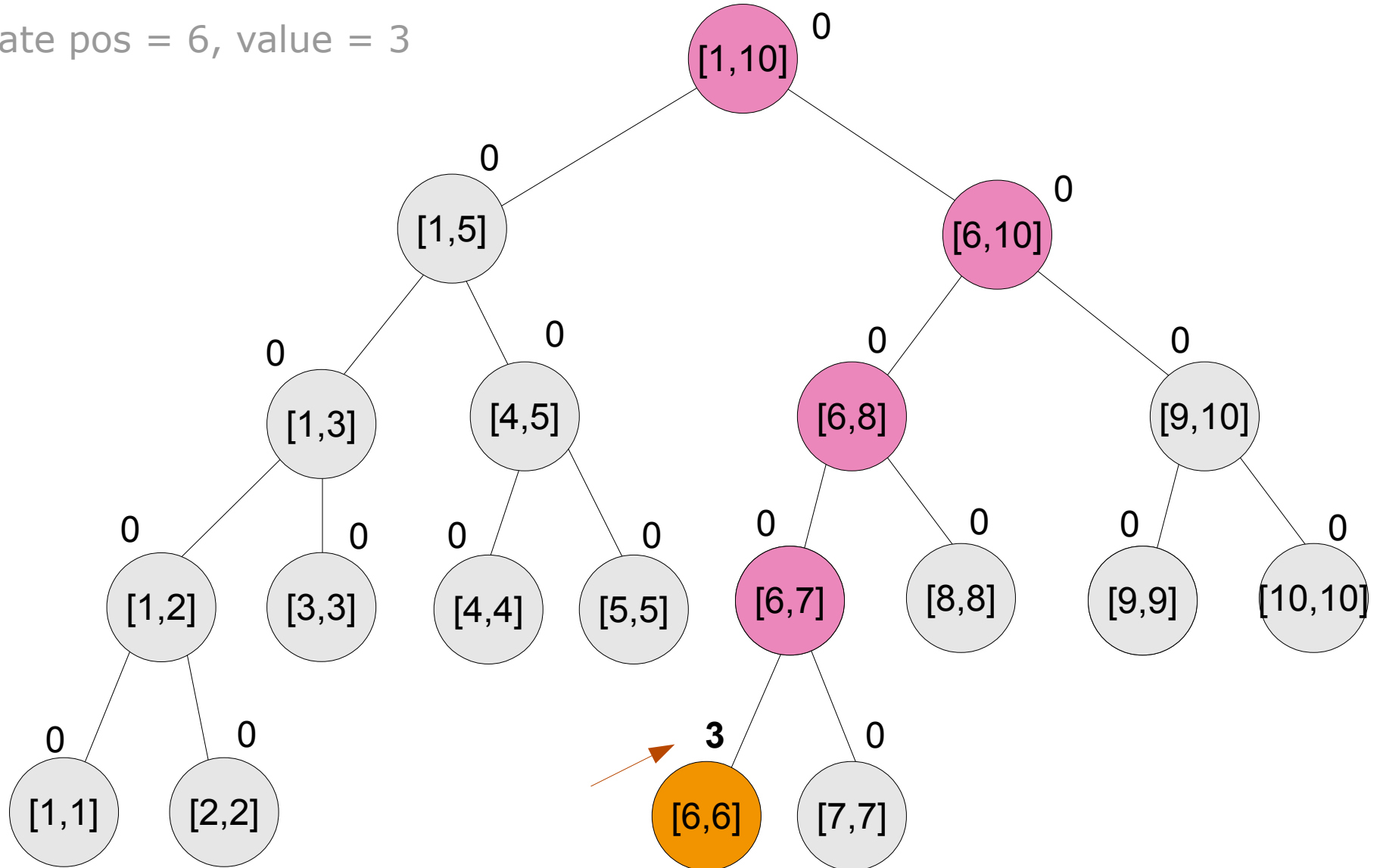
# Interval Trees

Update pos = 6, value = 3



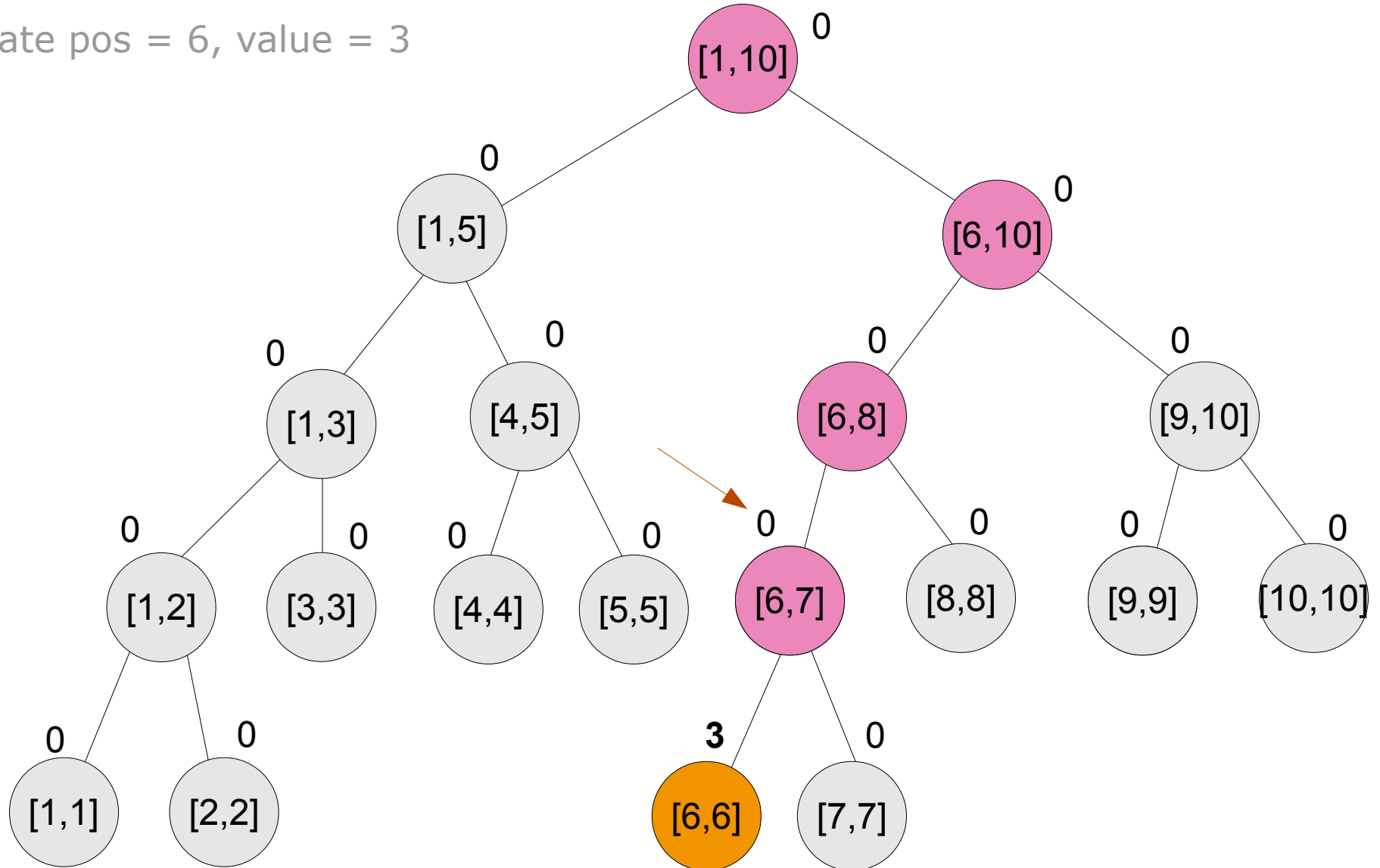
# Interval Trees

Update pos = 6, value = 3



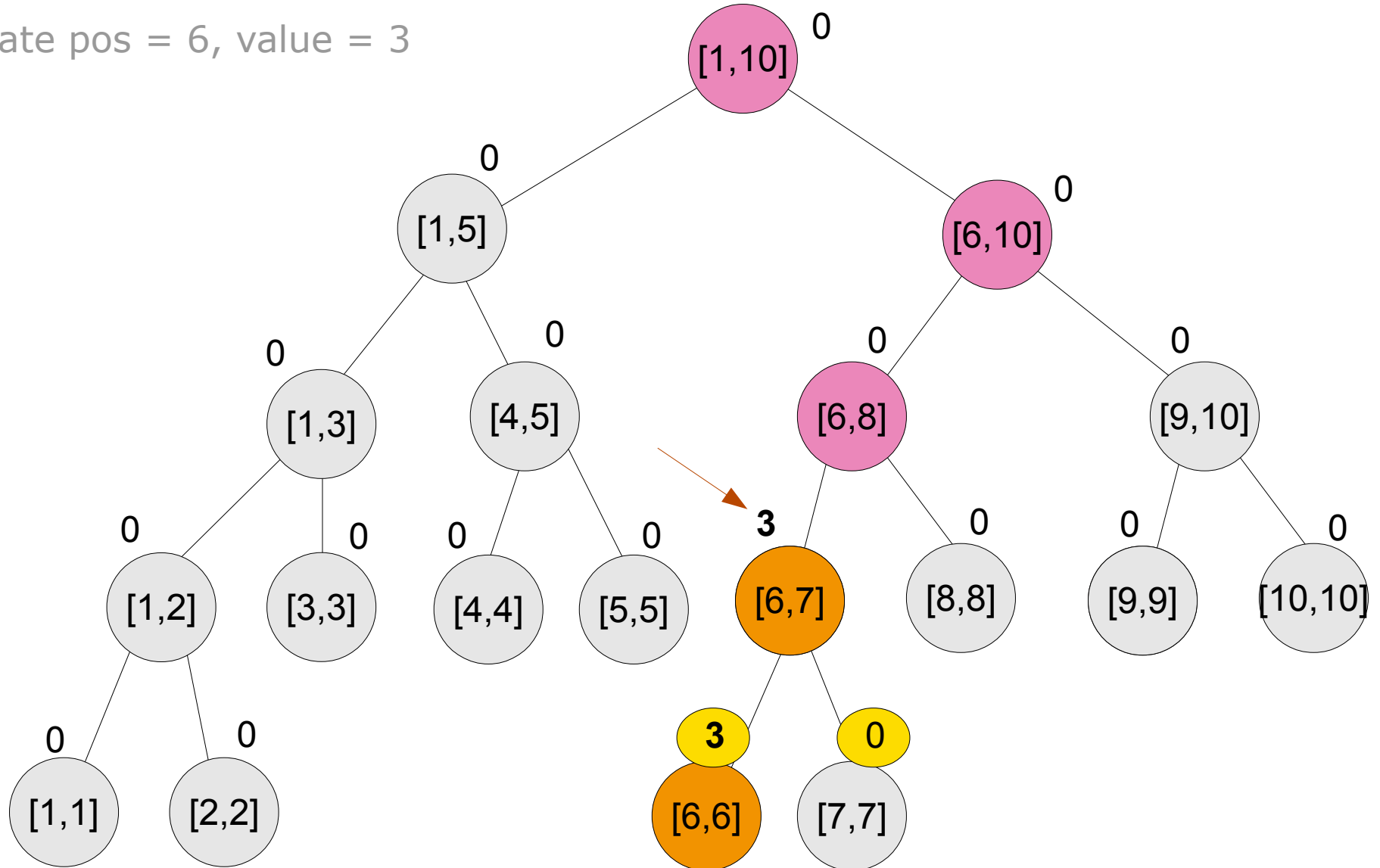
# Interval Trees

Update pos = 6, value = 3



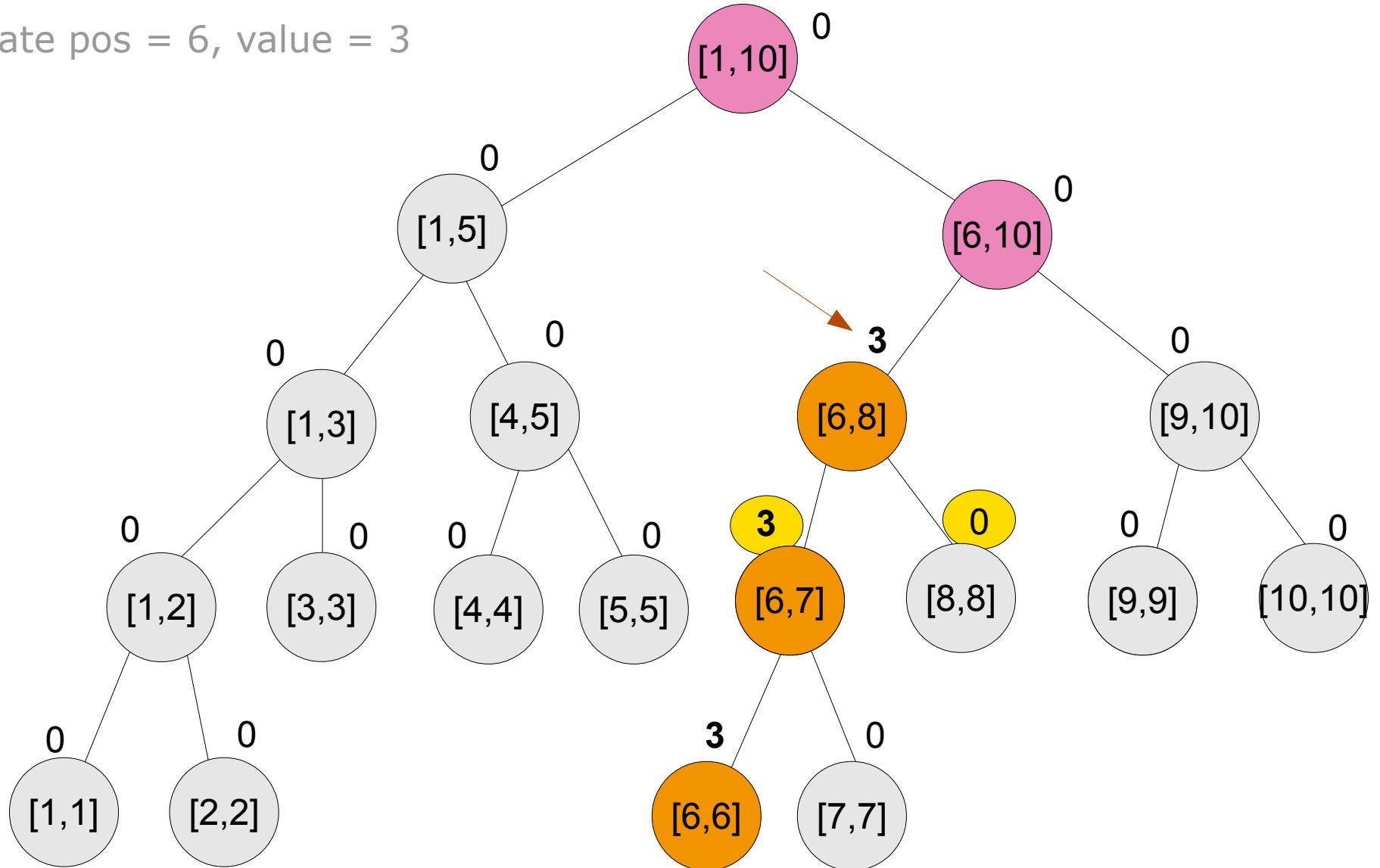
# Interval Trees

Update pos = 6, value = 3



# Interval Trees

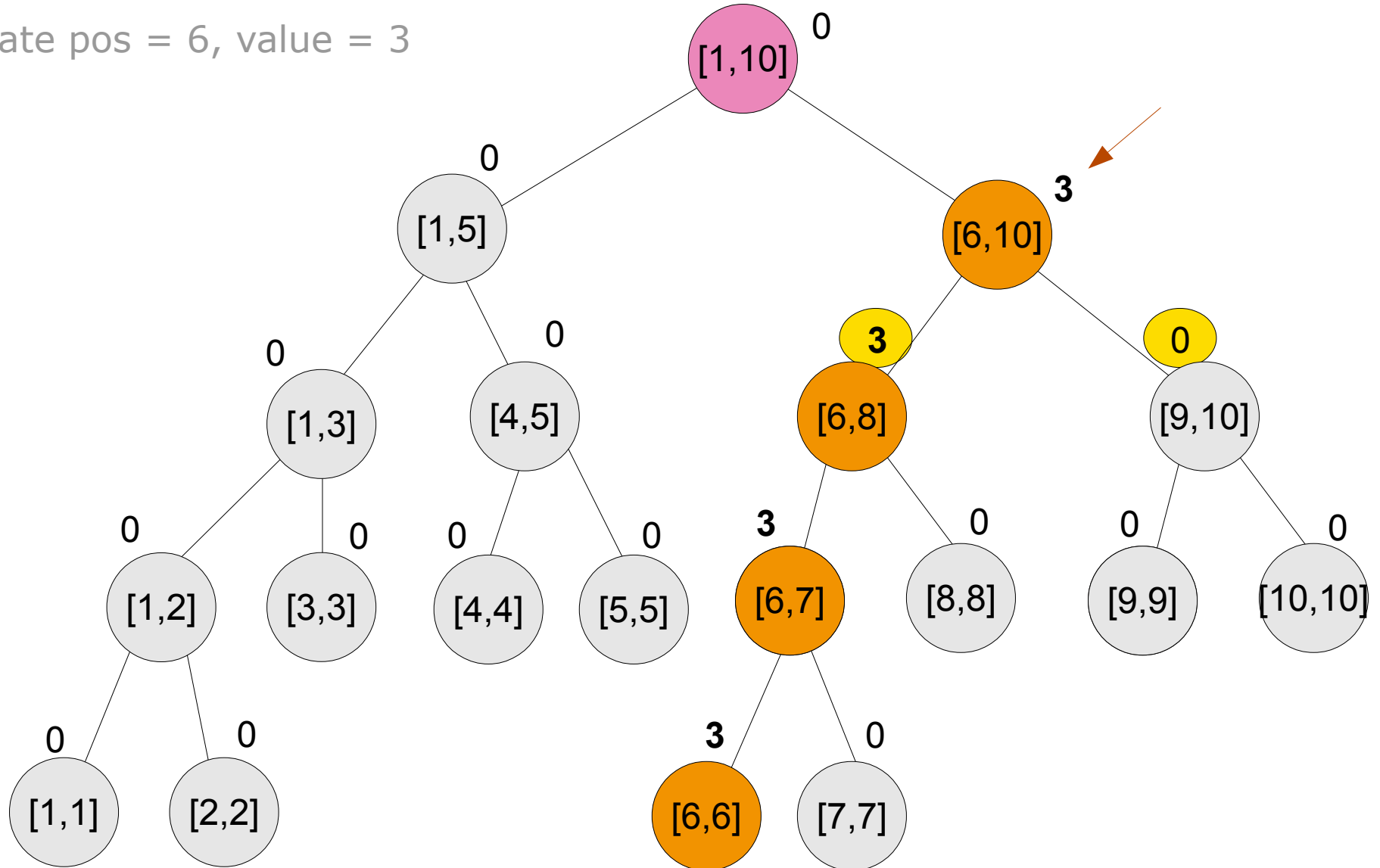
Update pos = 6, value = 3





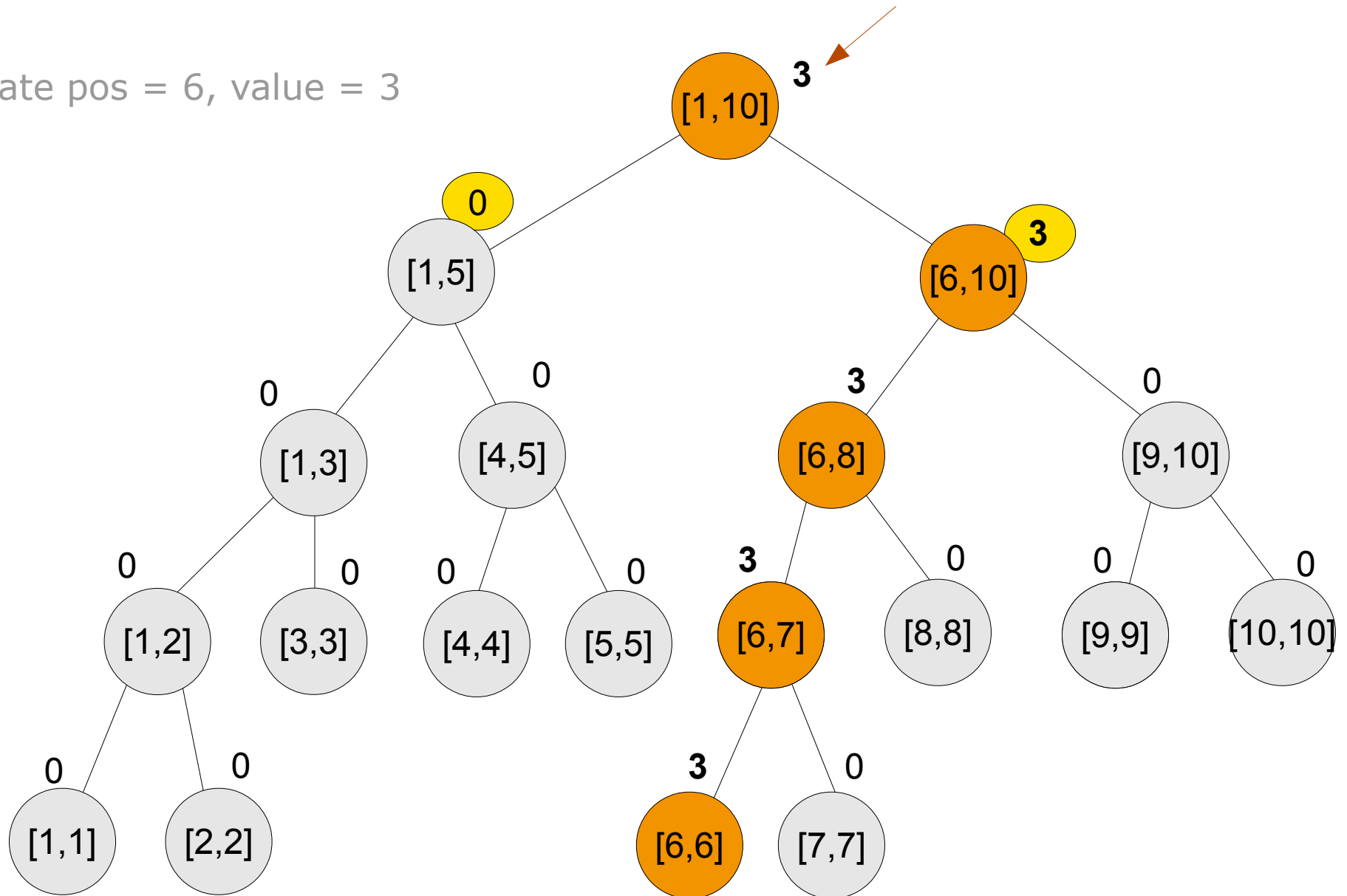
# Interval Trees

Update pos = 6, value = 3



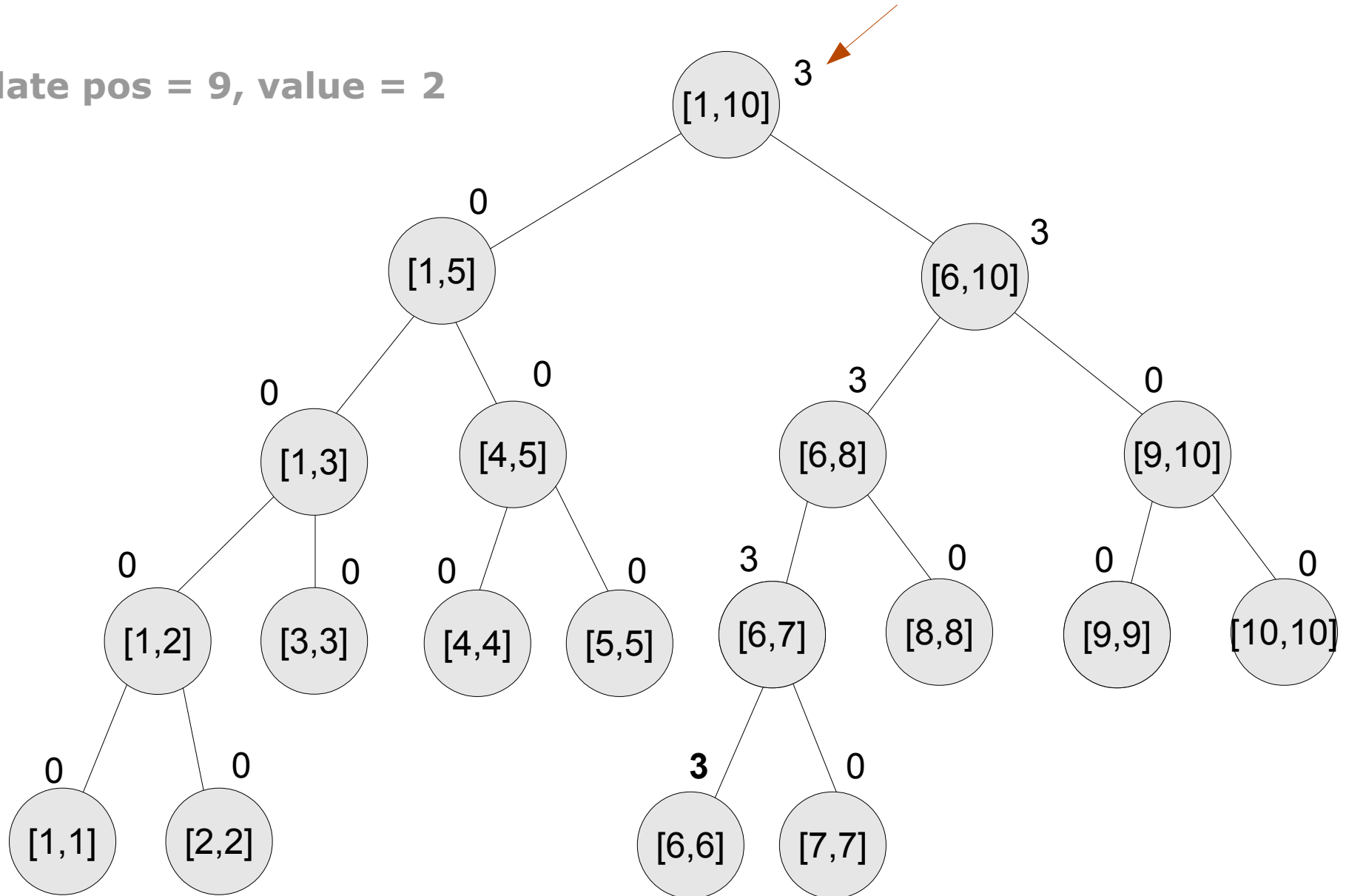
# Interval Trees

Update pos = 6, value = 3



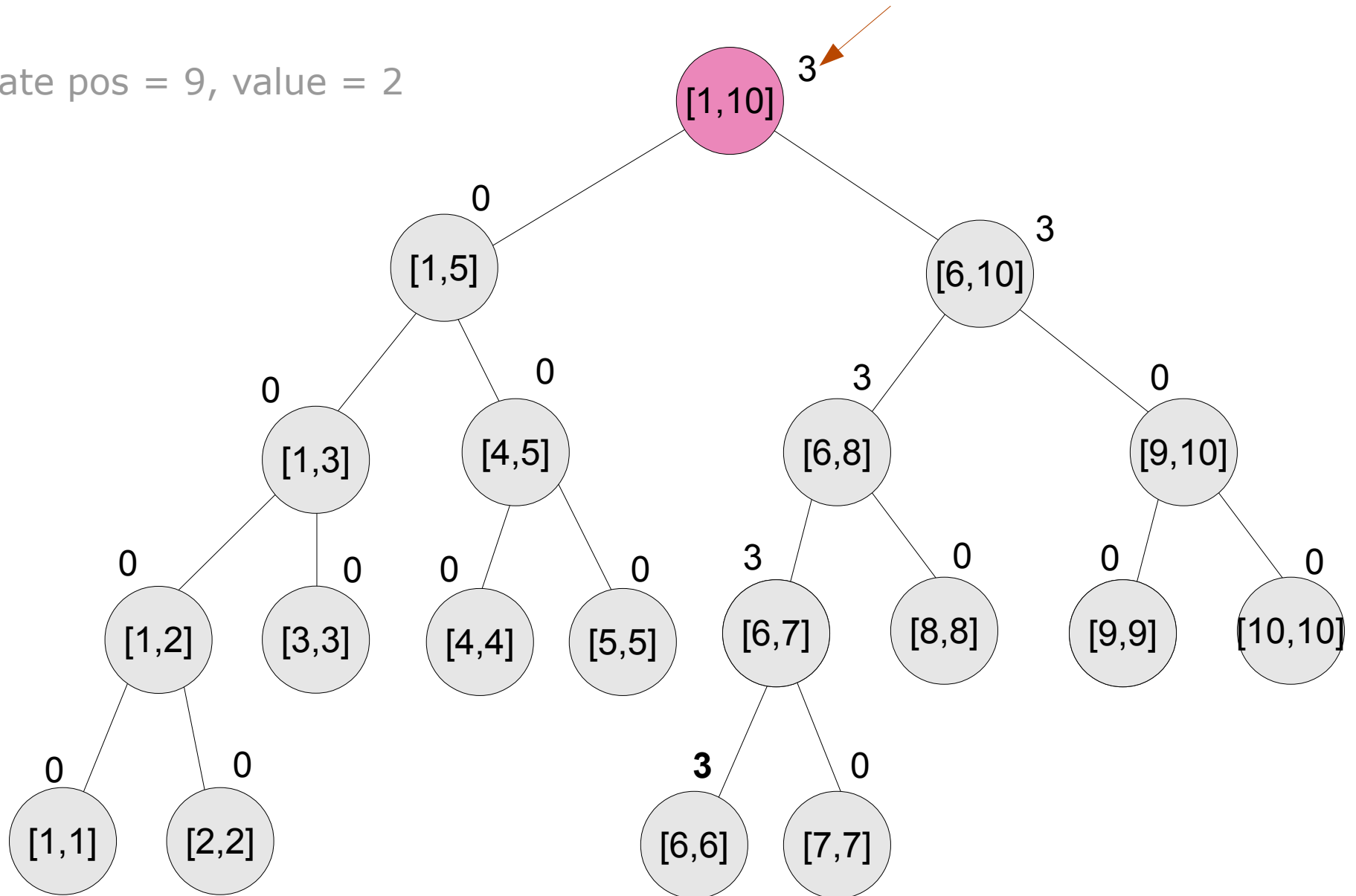
# Interval Trees

Update pos = 9, value = 2



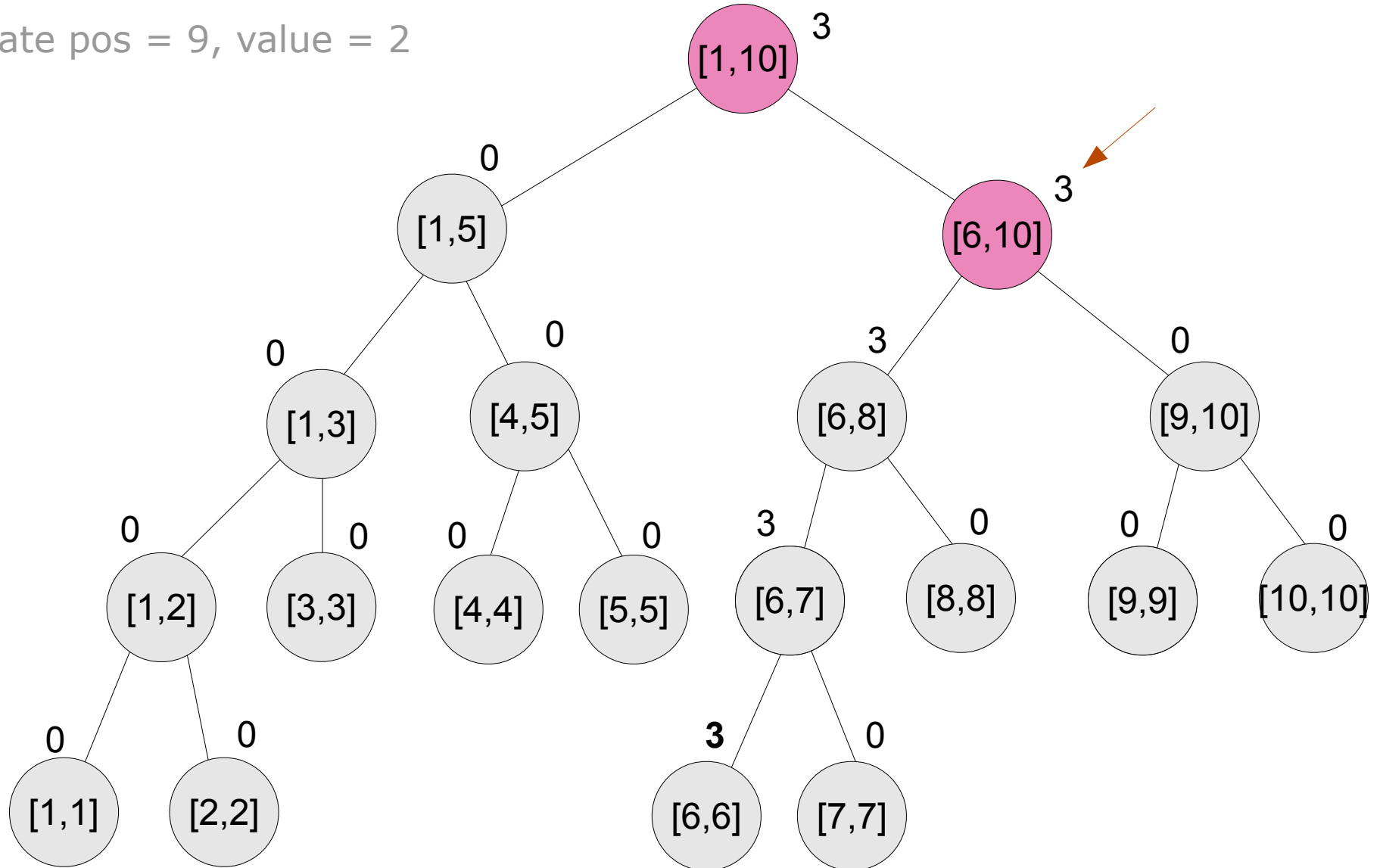
# Interval Trees

Update pos = 9, value = 2



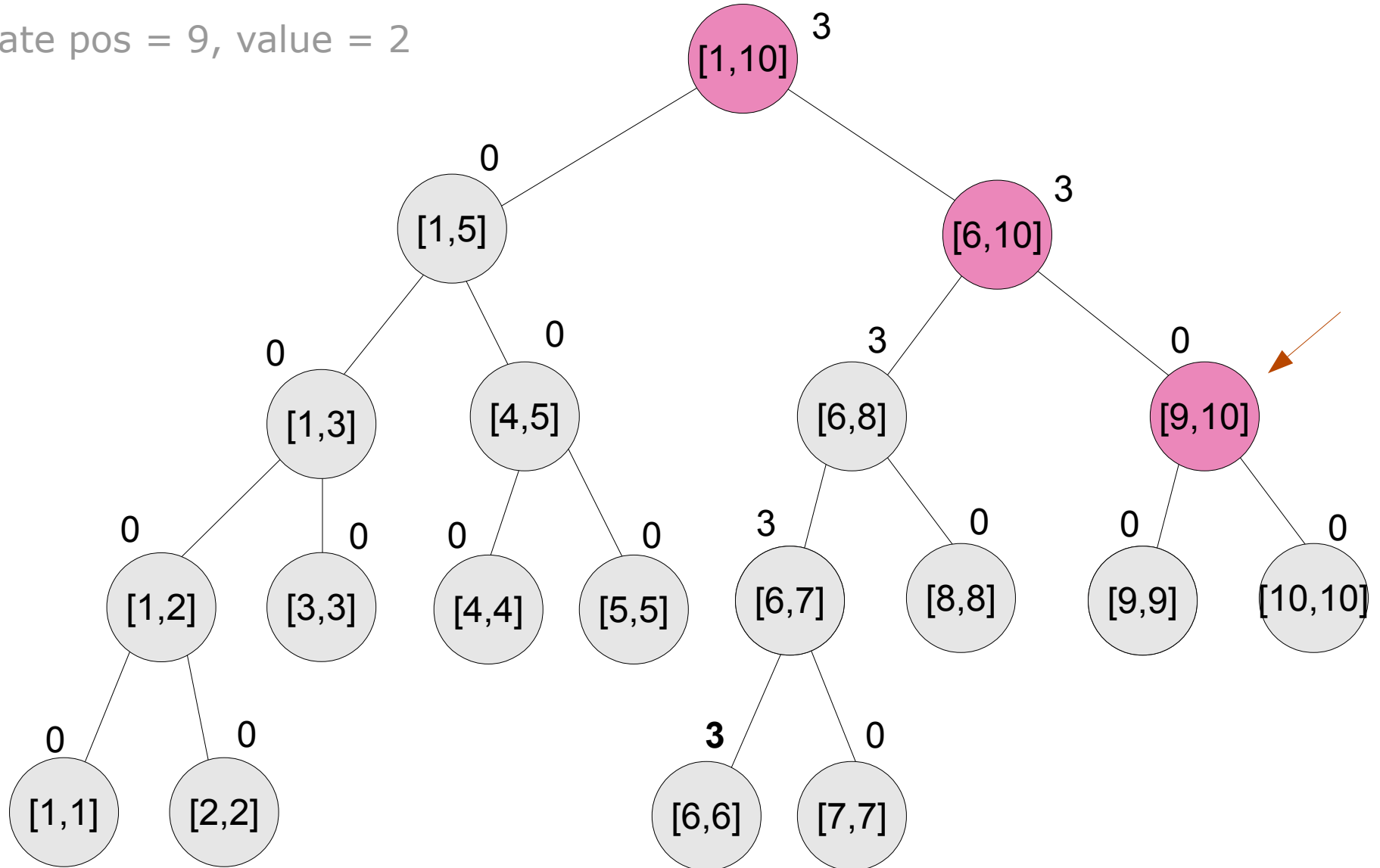
# Interval Trees

Update pos = 9, value = 2



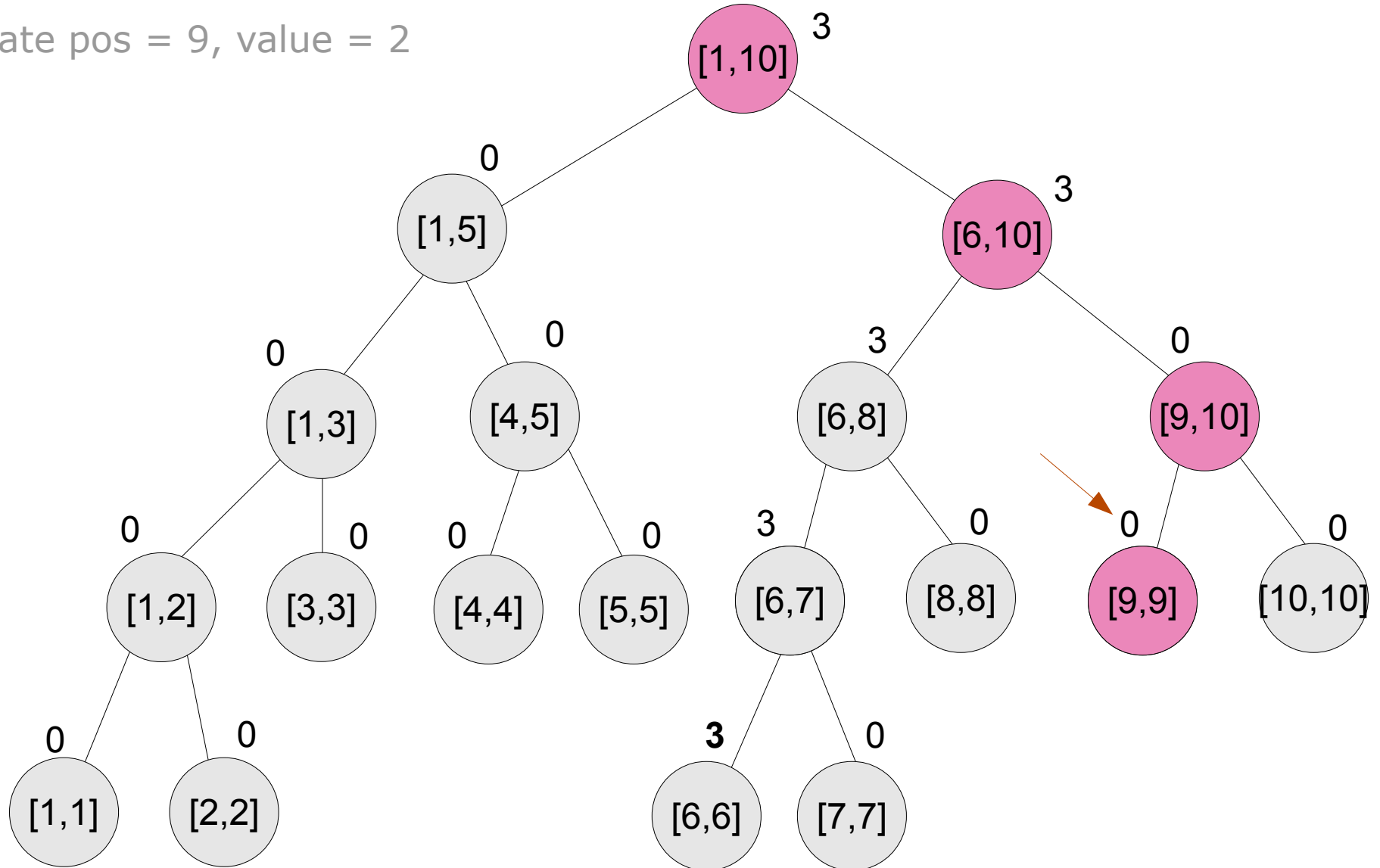
# Interval Trees

Update pos = 9, value = 2



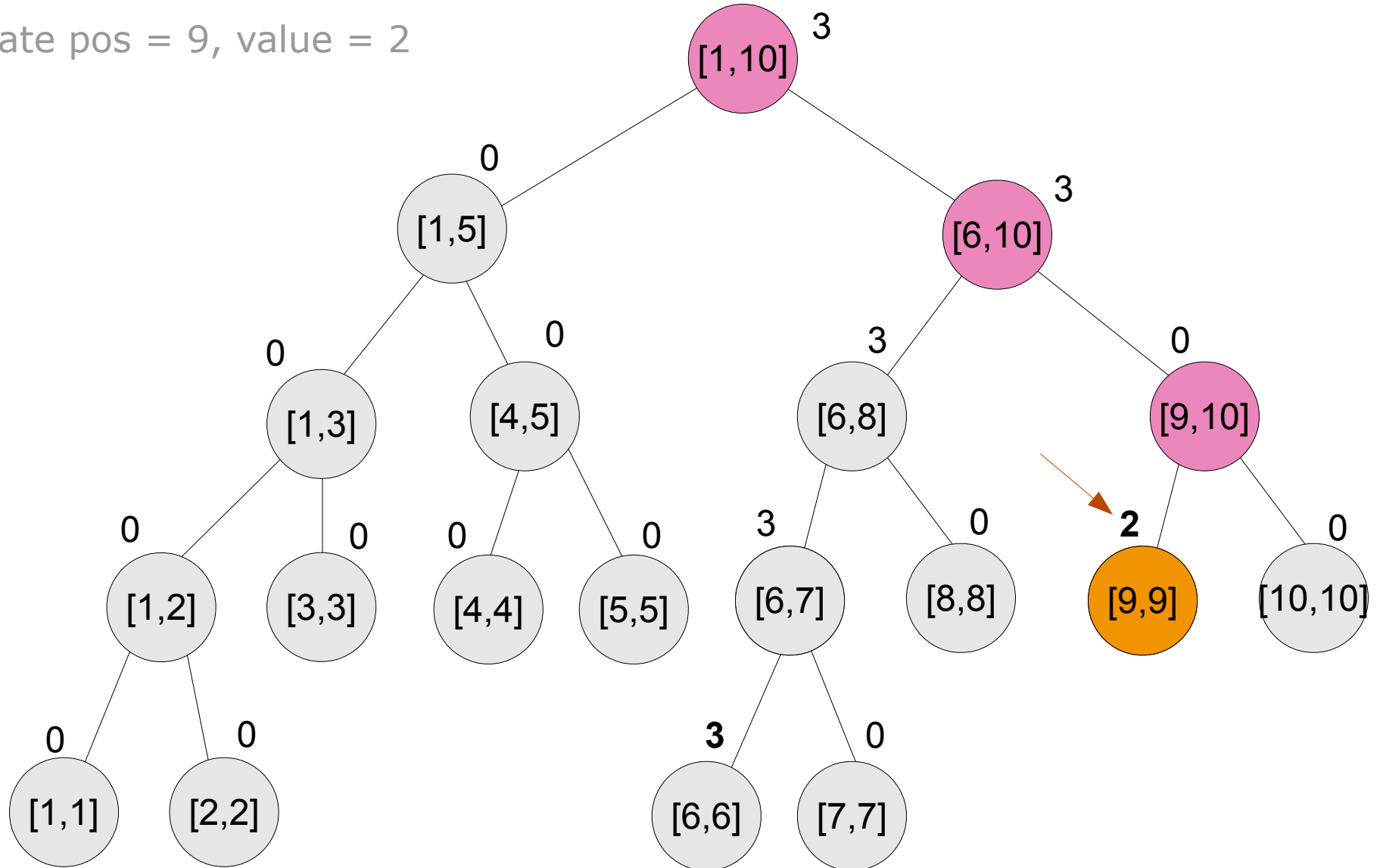
# Interval Trees

Update pos = 9, value = 2



# Interval Trees

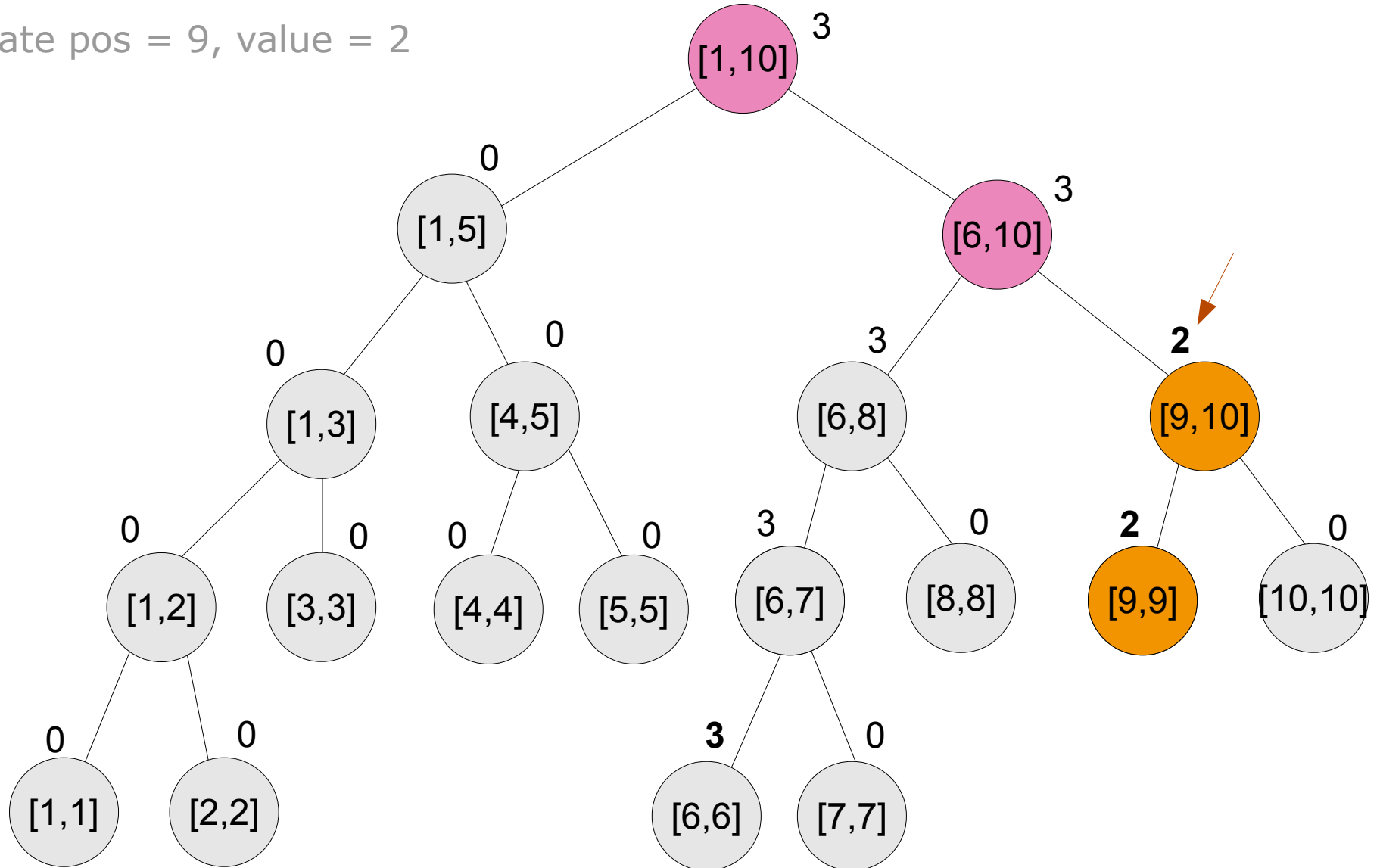
Update pos = 9, value = 2





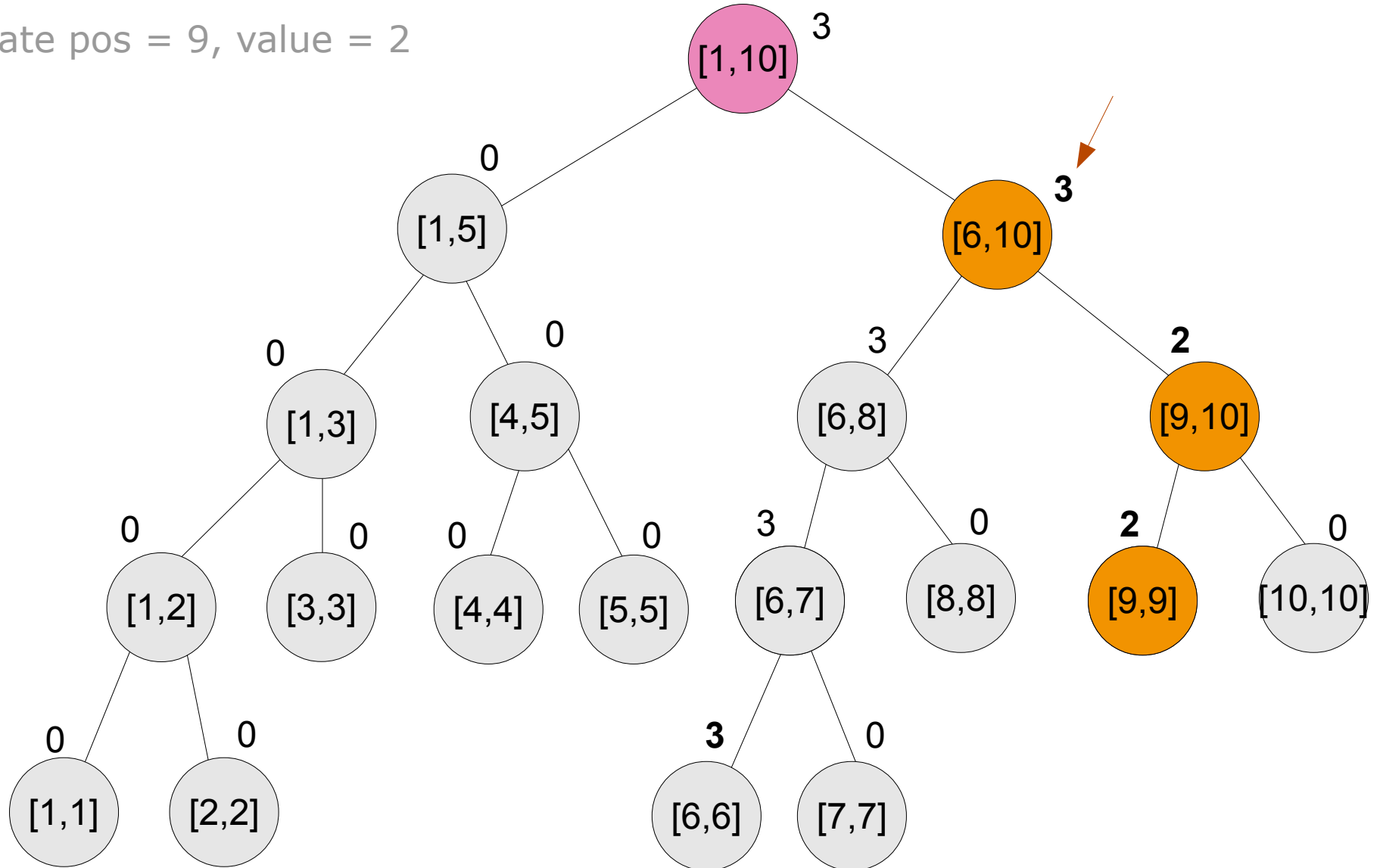
# Interval Trees

Update pos = 9, value = 2



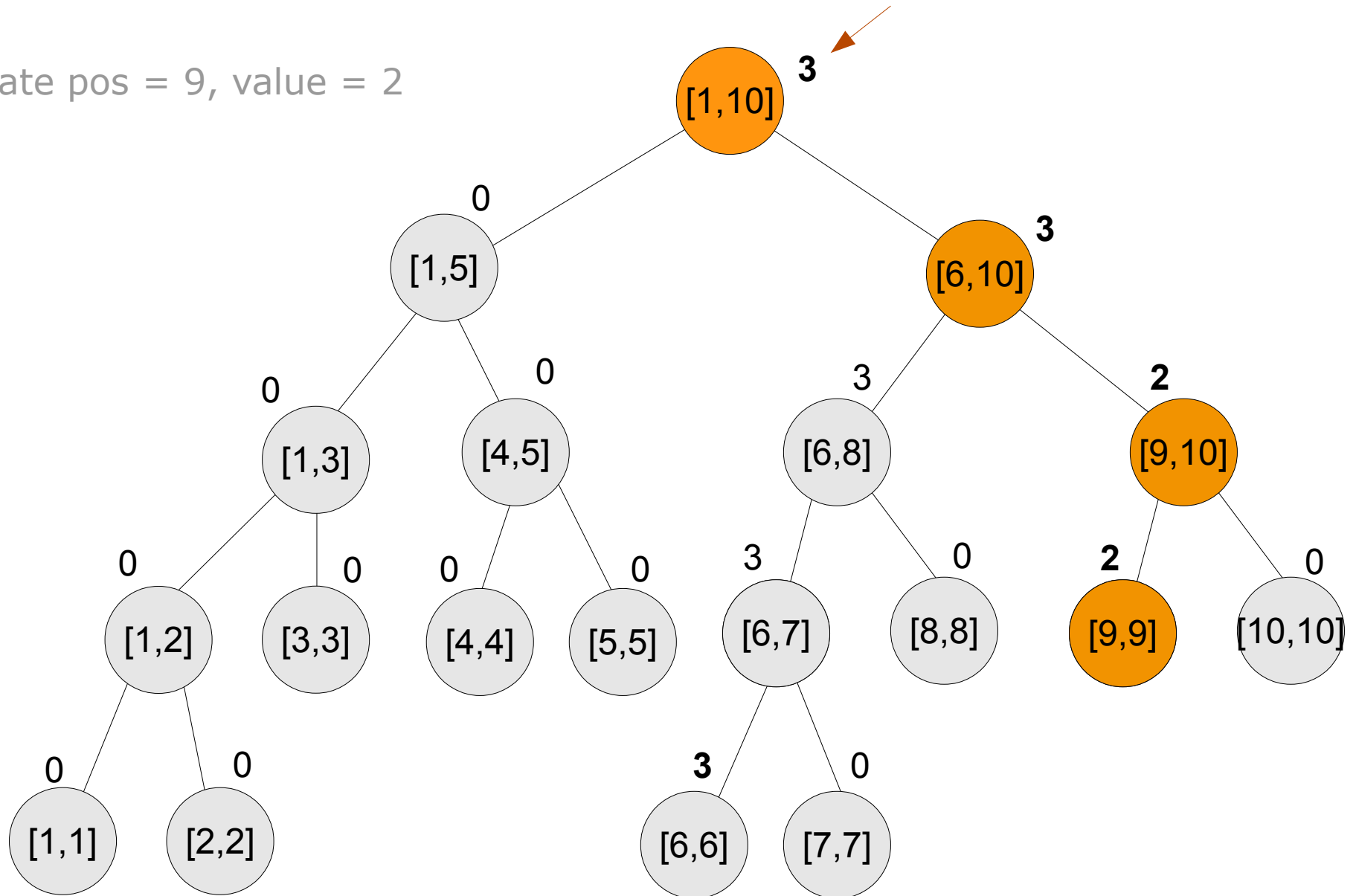
# Interval Trees

Update pos = 9, value = 2



# Interval Trees

Update pos = 9, value = 2



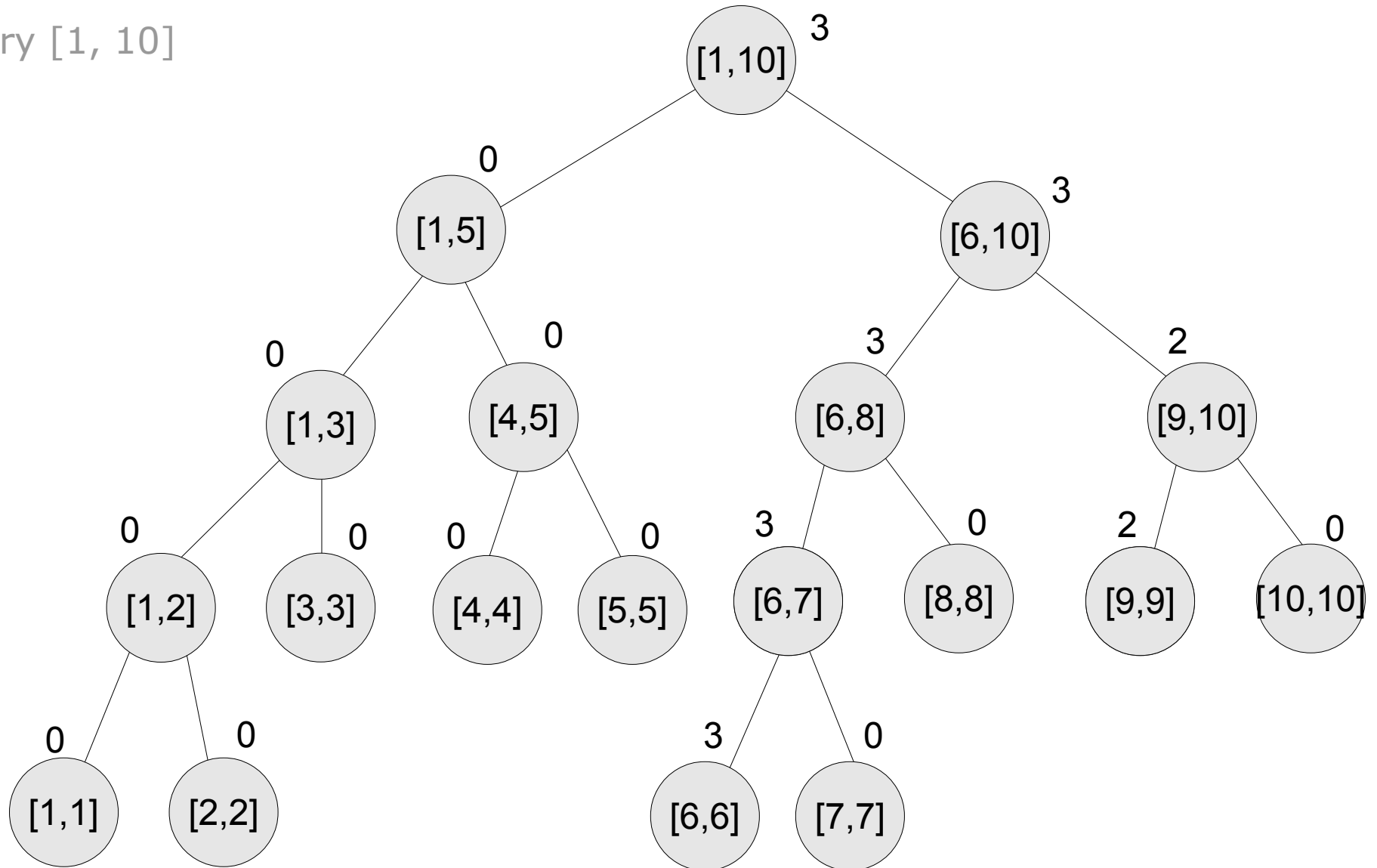
# Interval Trees

```
int update(int pos, int value, int x, int y, int id) {  
    int mid;  
    if ( x == y ) {  
        T[id] = value;  
    }  
    else {  
        mid = (x + y)/2;  
        if ( pos <= mid ) {  
            update(pos, value, x, mid, 2*id);  
        }  
        else {  
            update(pos, value, mid+1, y, 2*id + 1);  
        }  
        T[id] = max(T[2*id], T[2*id + 1]);  
    }  
}
```

```
update(pos, value, 1, N, 1);
```

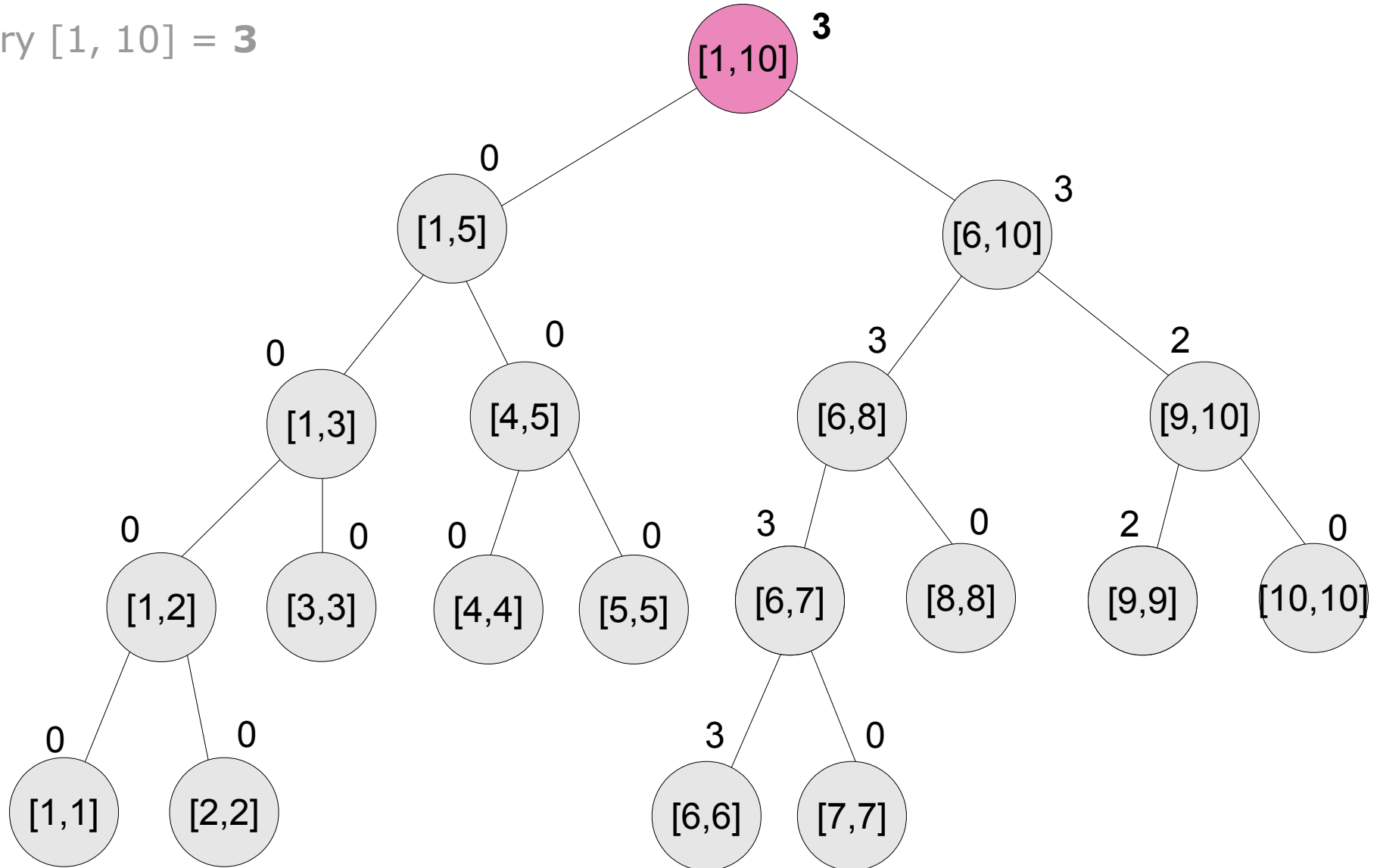
# Interval Trees

Query [1, 10]



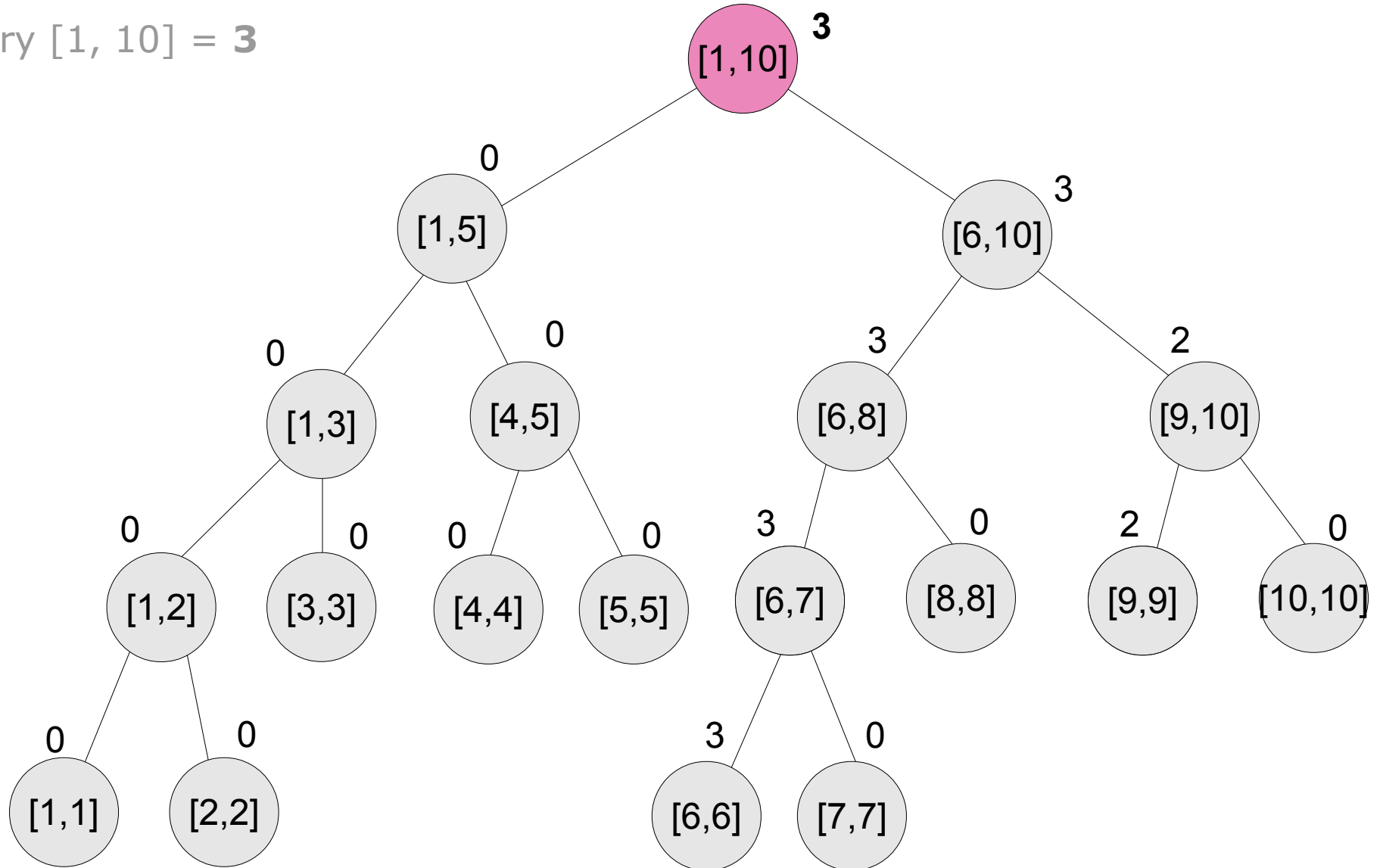
# Interval Trees

Query  $[1, 10] = 3$



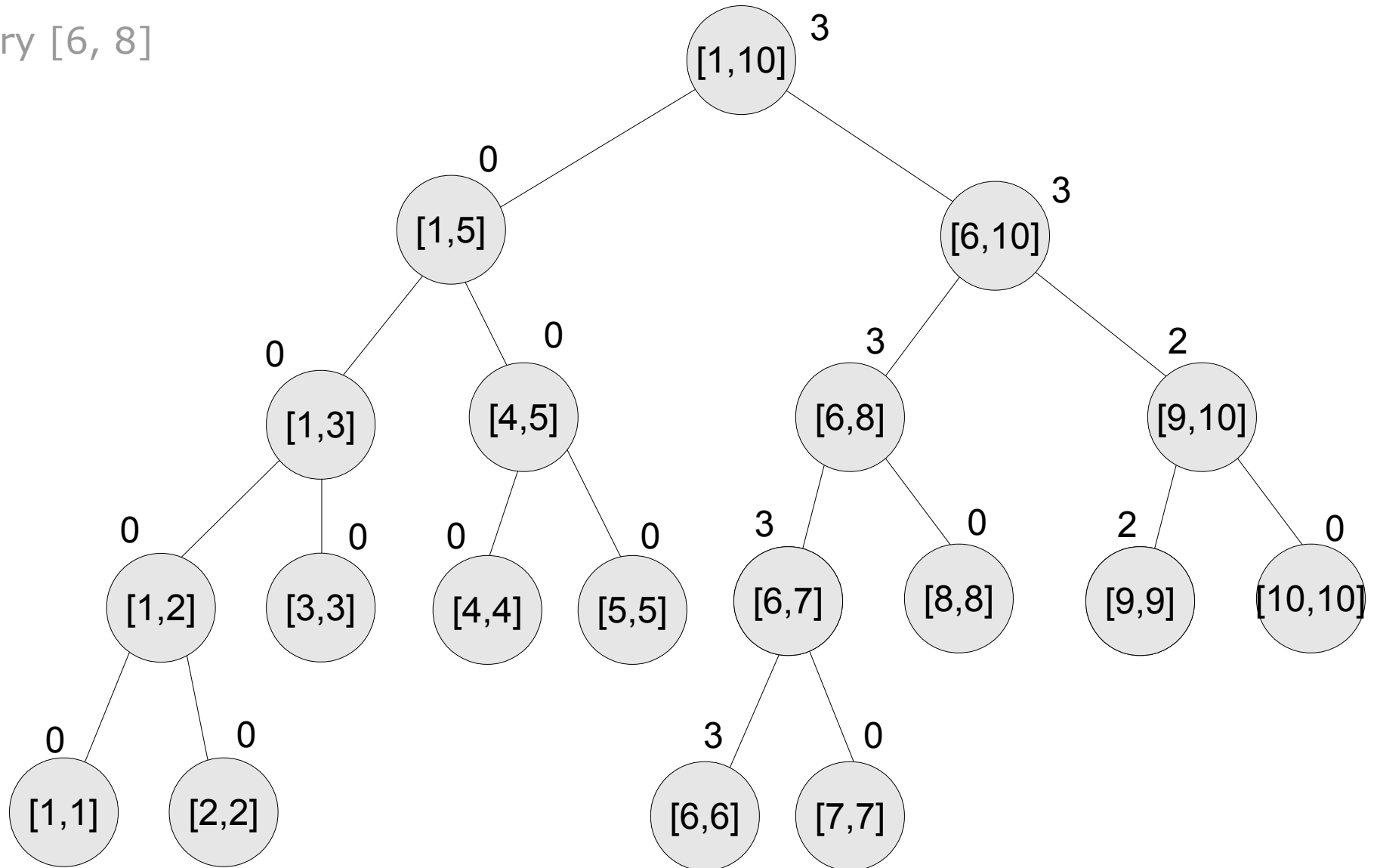
# Interval Trees

Query  $[1, 10] = 3$



# Interval Trees

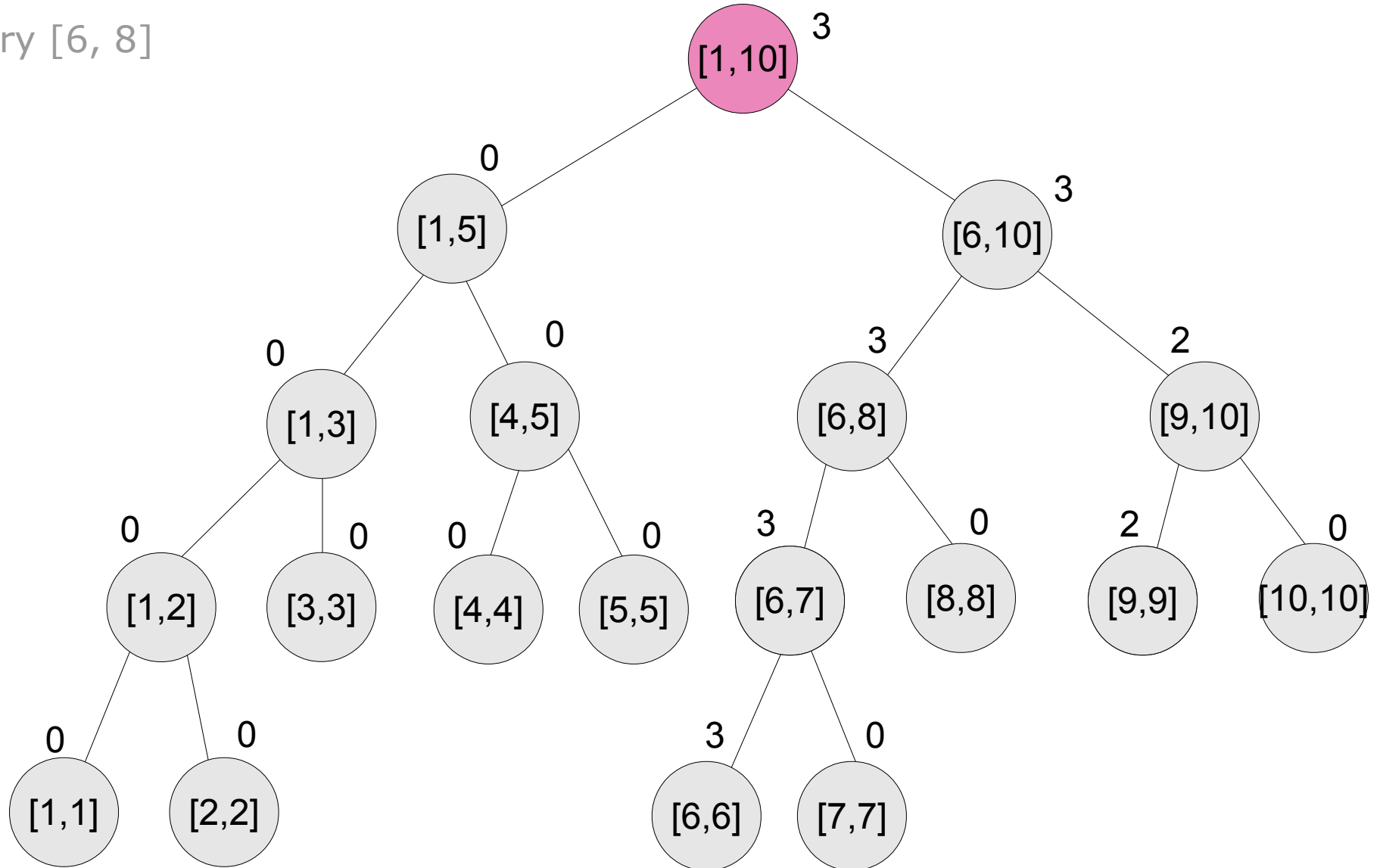
Query [6, 8]





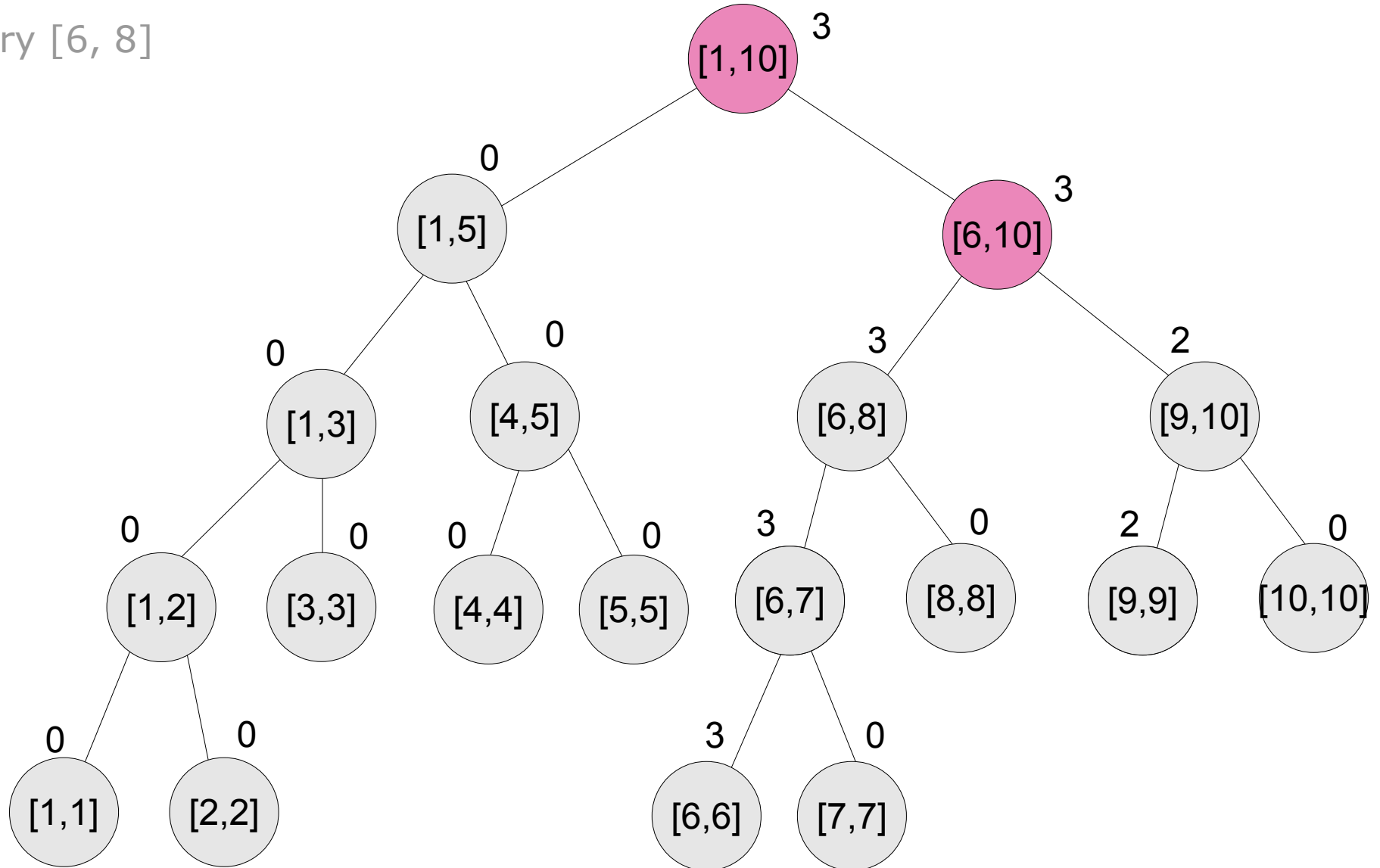
# Interval Trees

Query [6, 8]



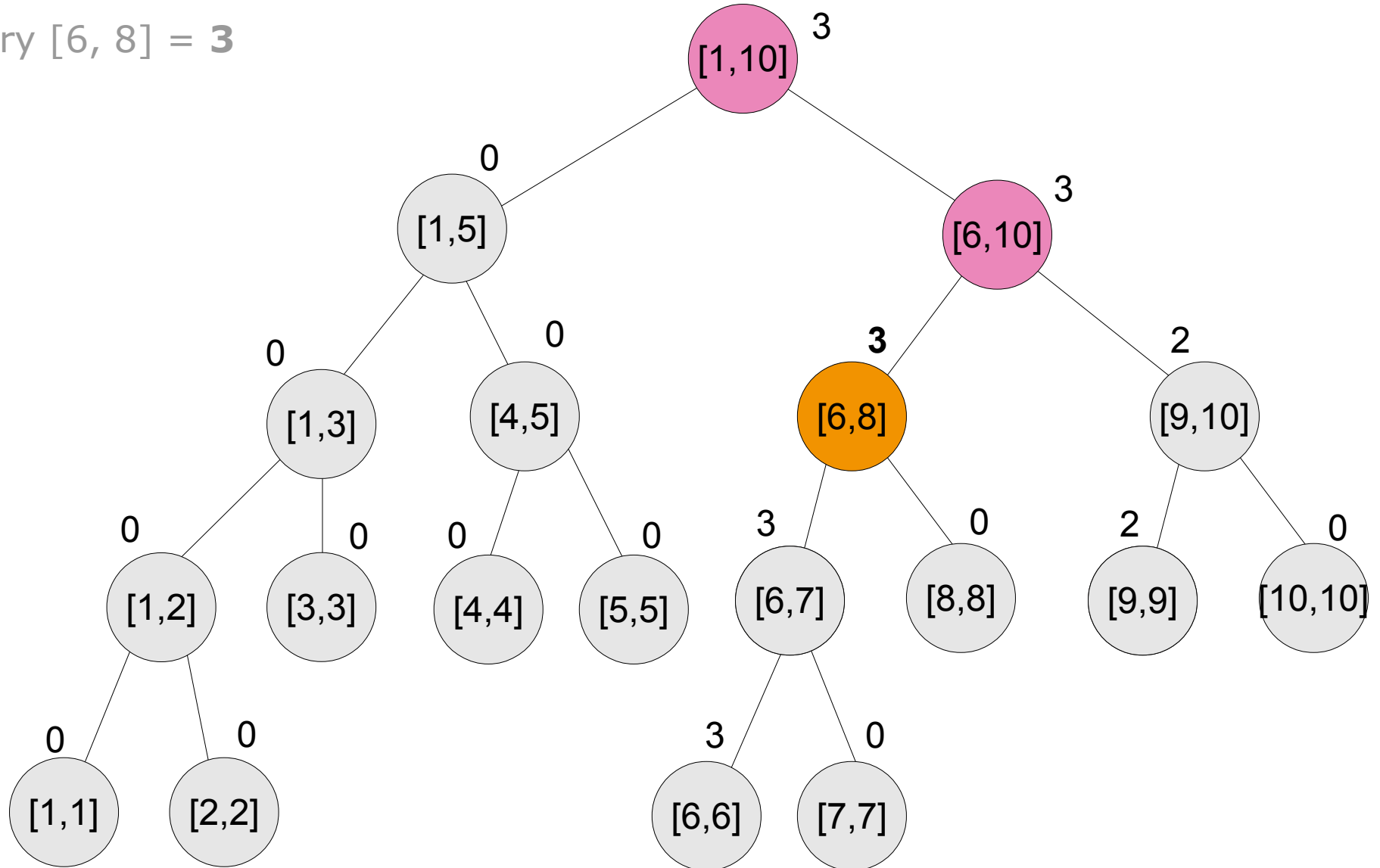
# Interval Trees

Query [6, 8]



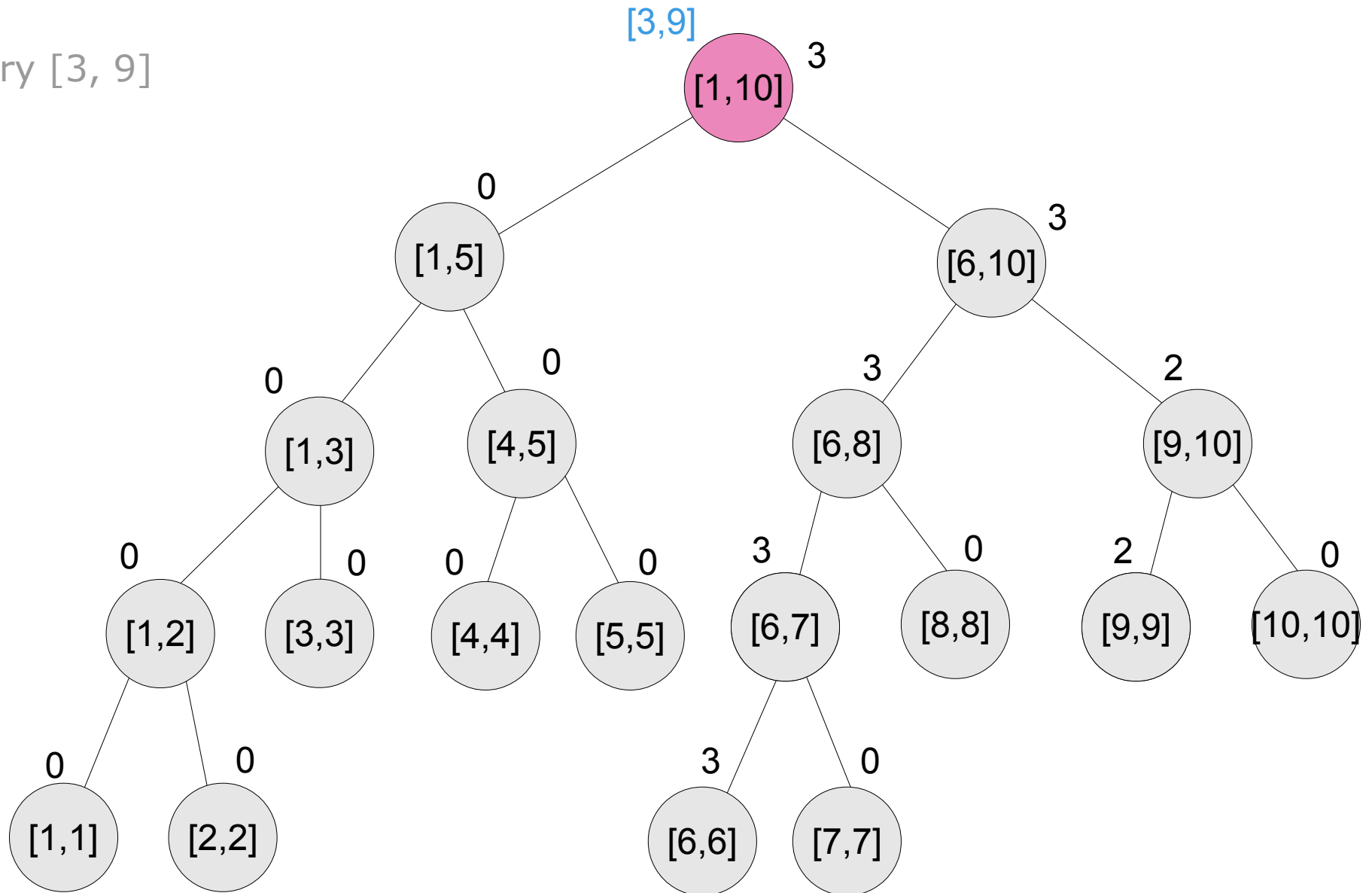
# Interval Trees

Query  $[6, 8] = \mathbf{3}$



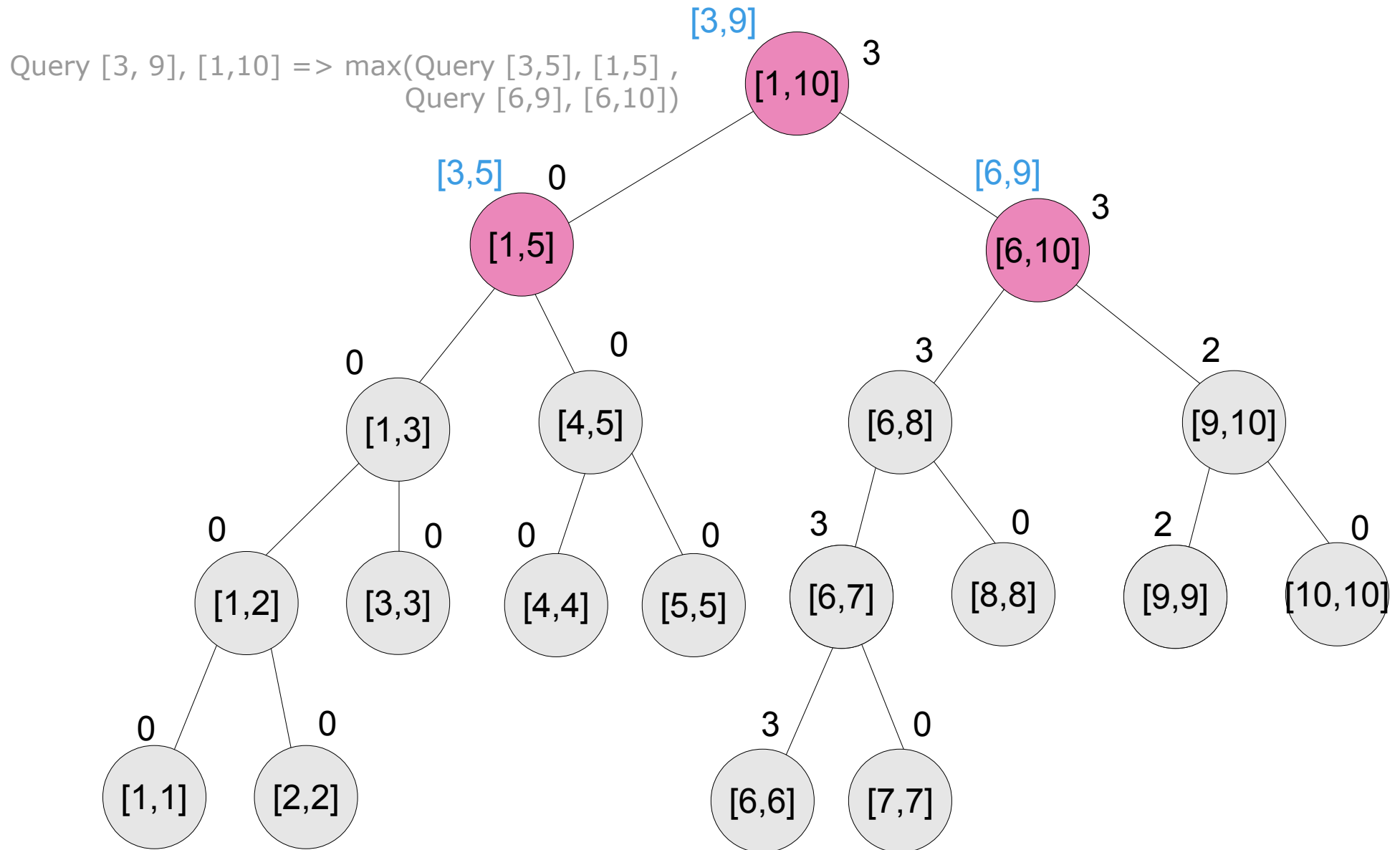
# Interval Trees

Query [3, 9]



Query [3, 9]

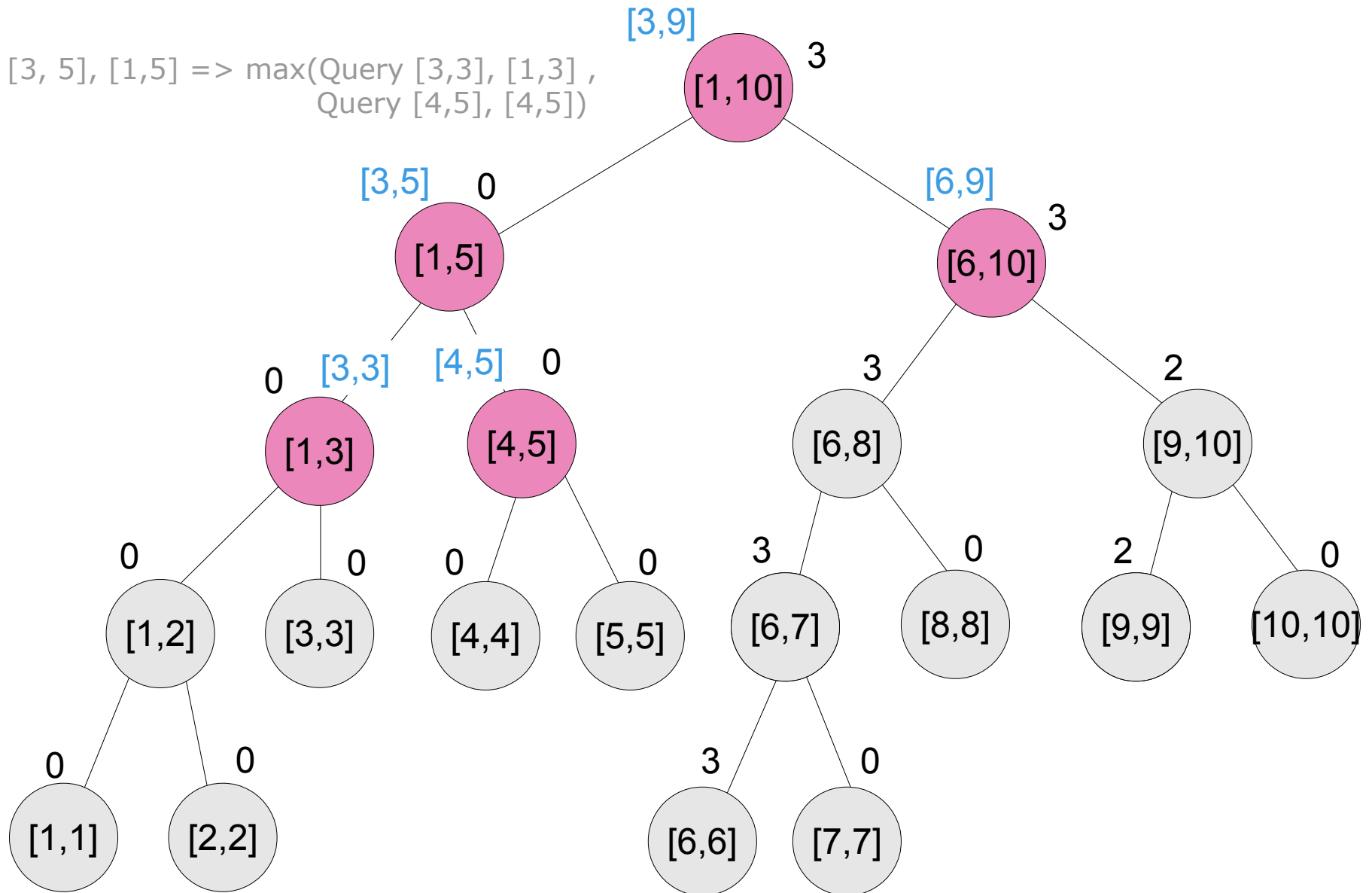
# Interval Trees



Query [3, 9]

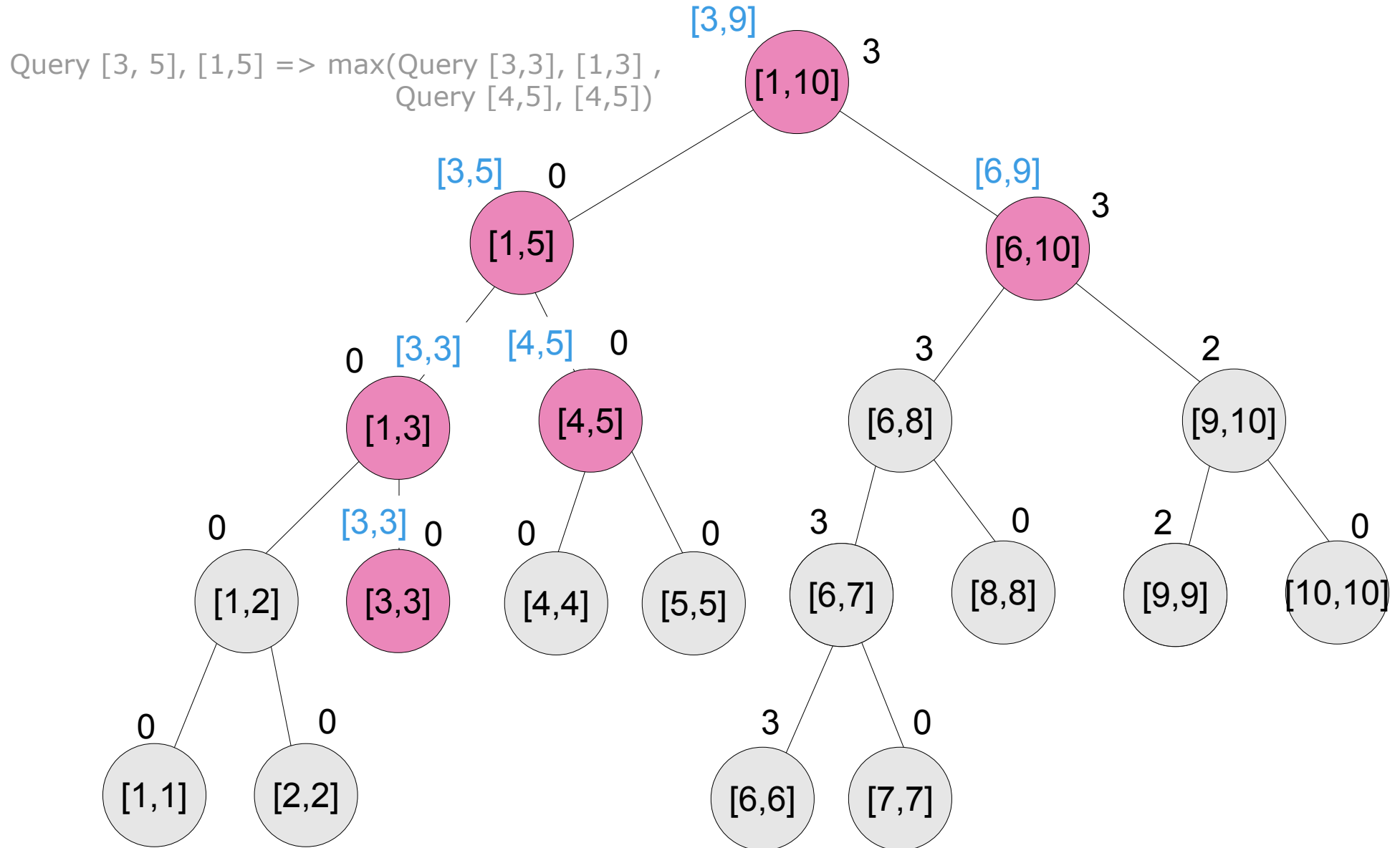
# Interval Trees

Query [3, 5], [1,5] => max(Query [3,3], [1,3] ,  
Query [4,5], [4,5])



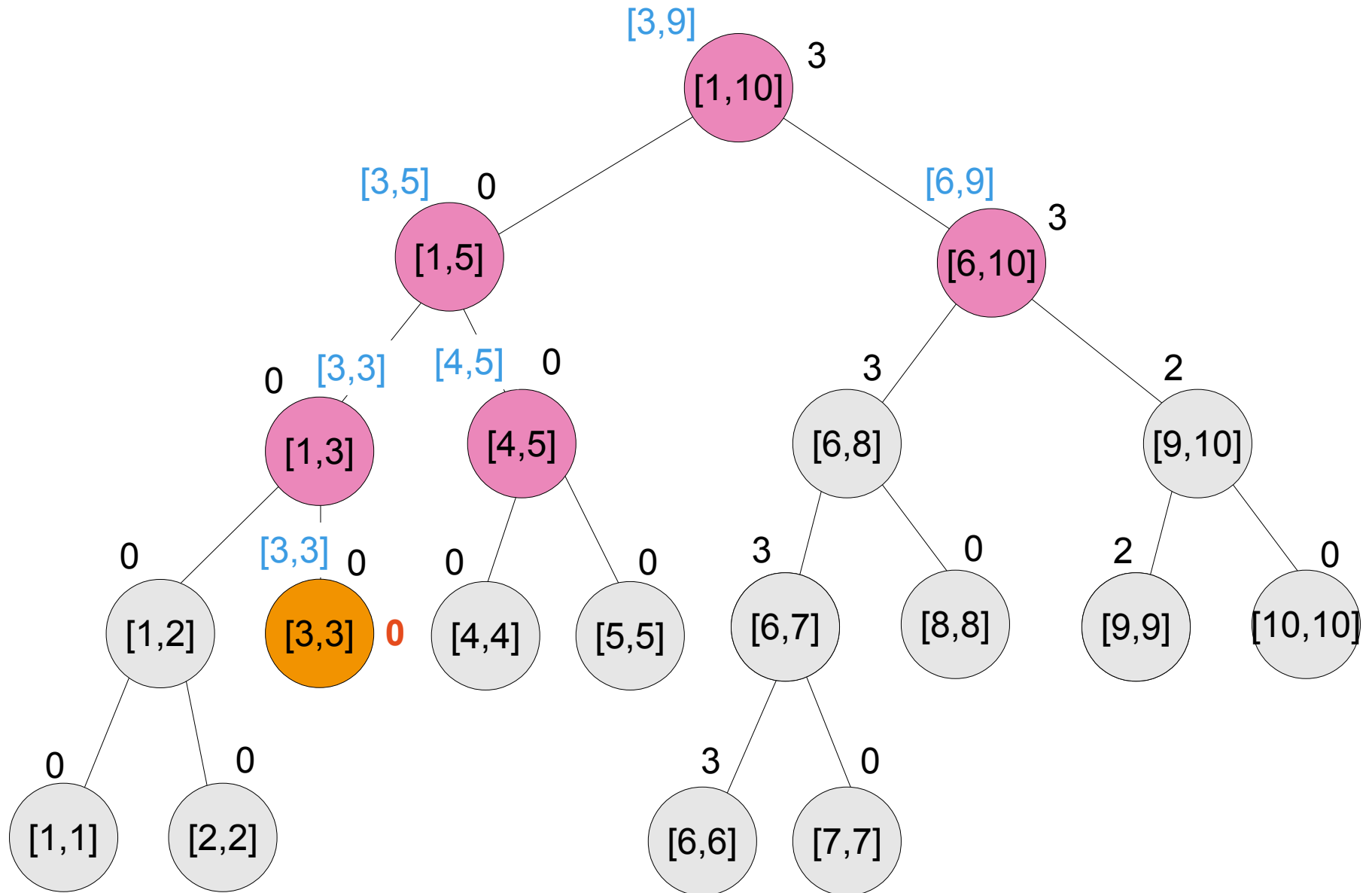
Query [3, 9]

# Interval Trees



Query [3, 9]

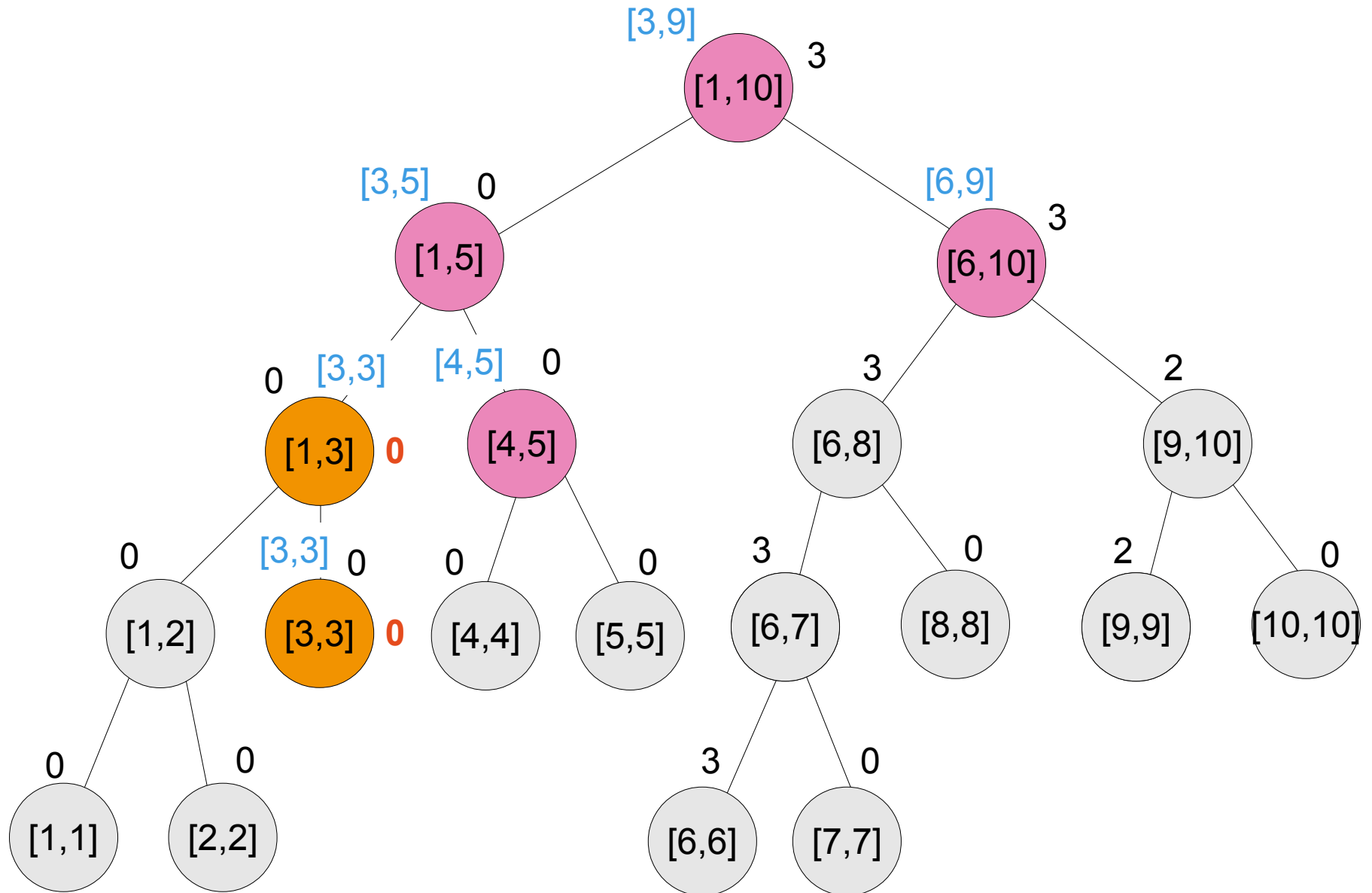
# Interval Trees





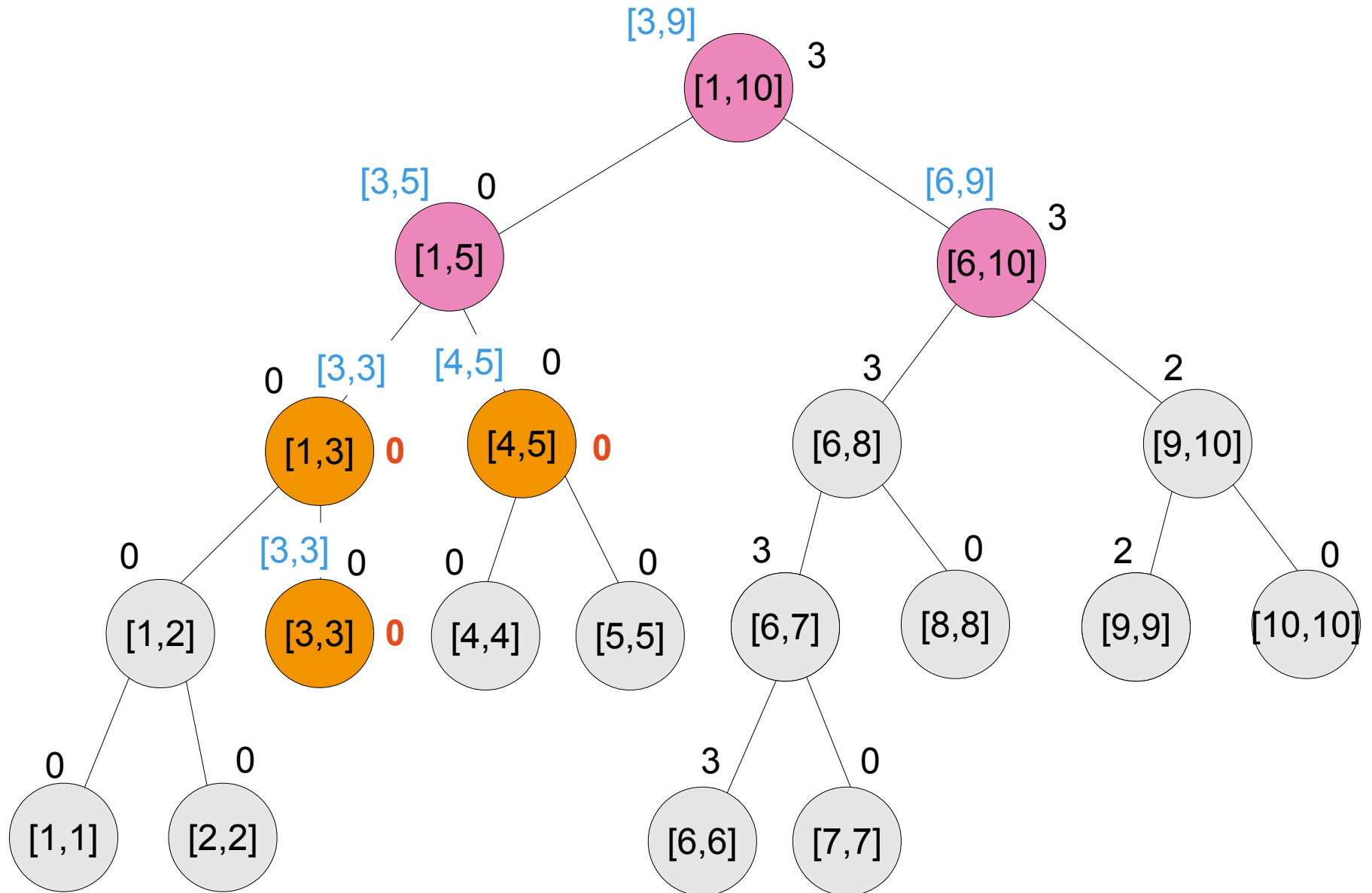
Query [3, 9]

# Interval Trees



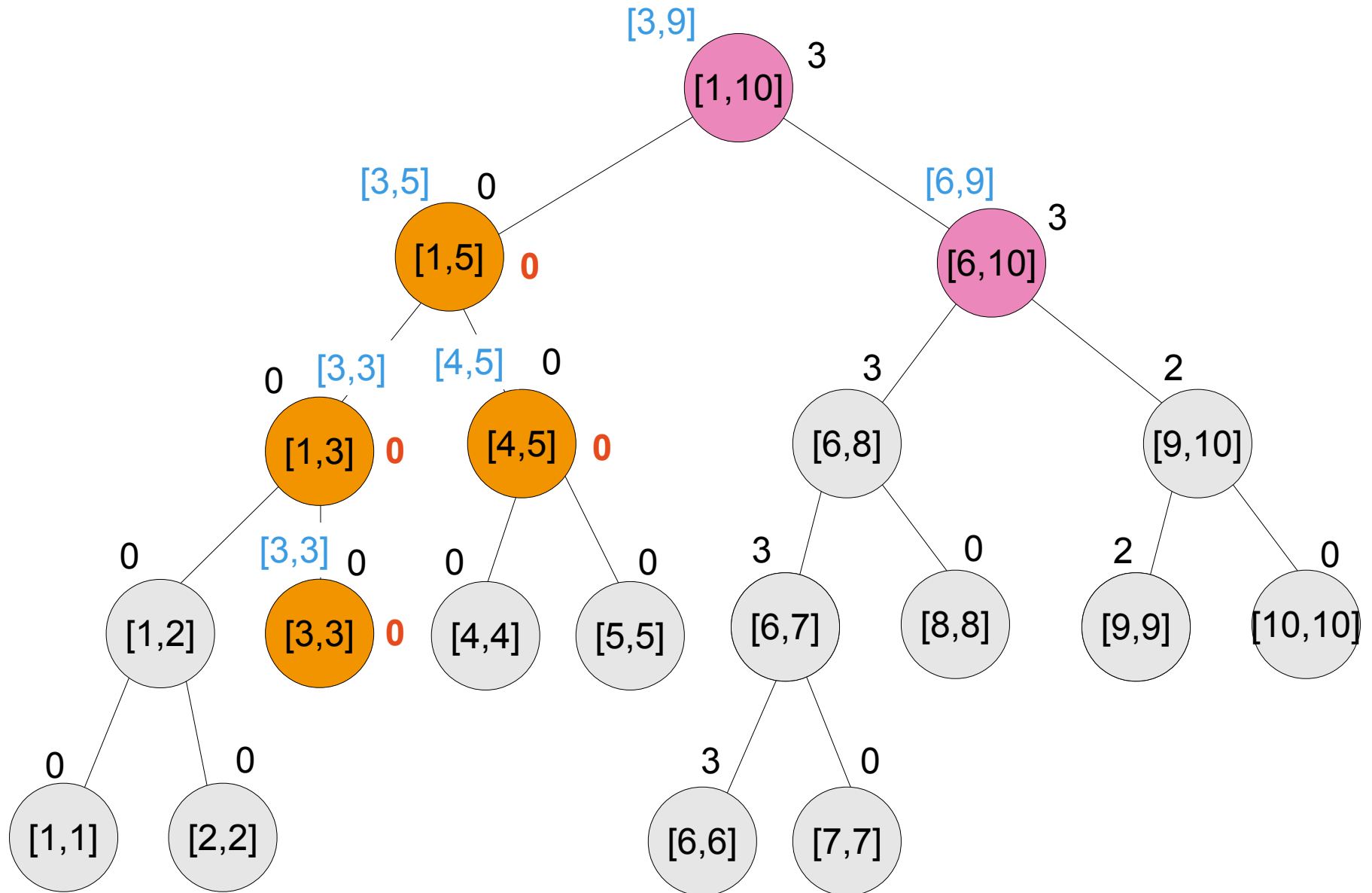
Query [3, 9]

# Interval Trees



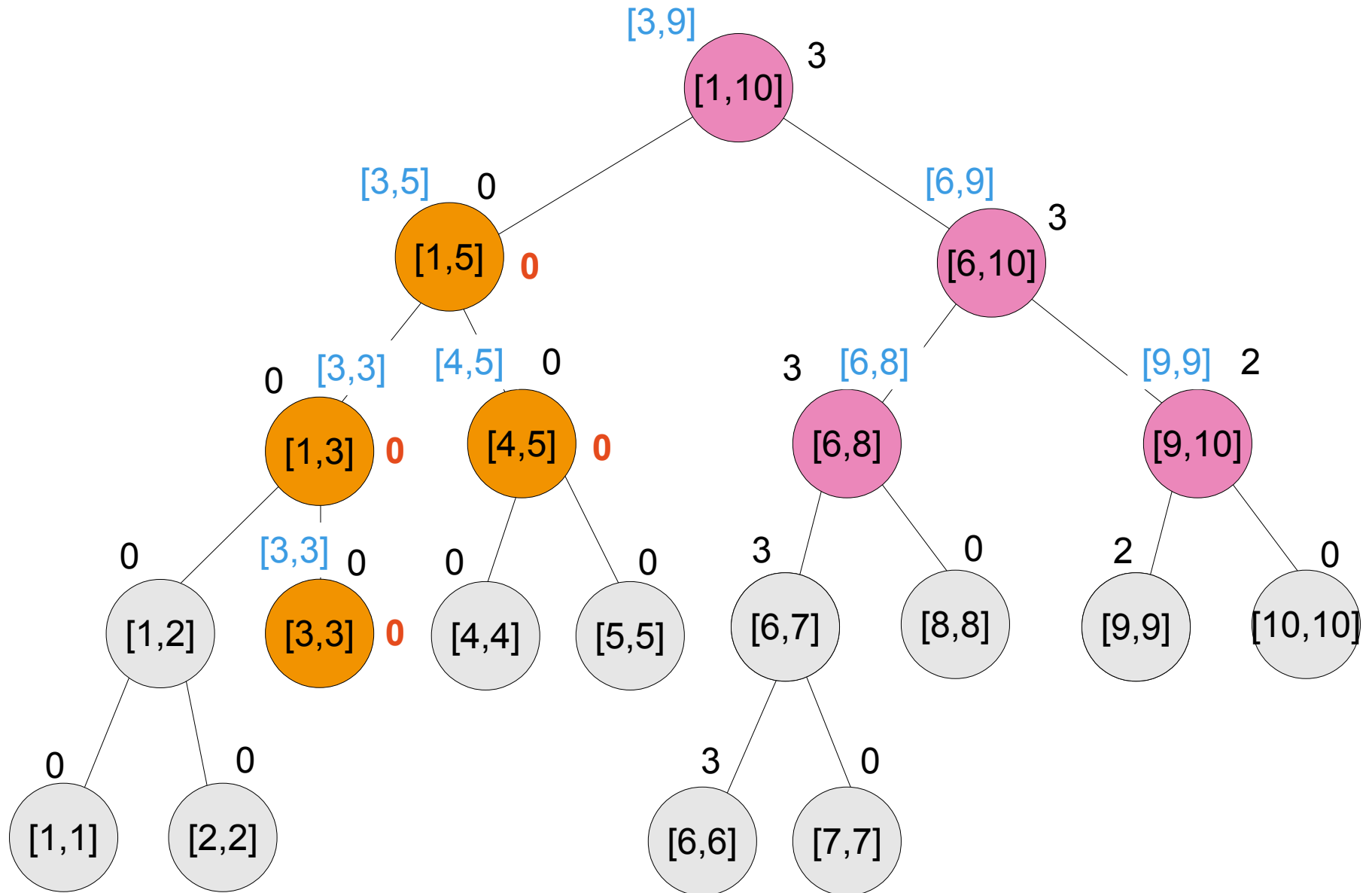
Query [3, 9]

# Interval Trees



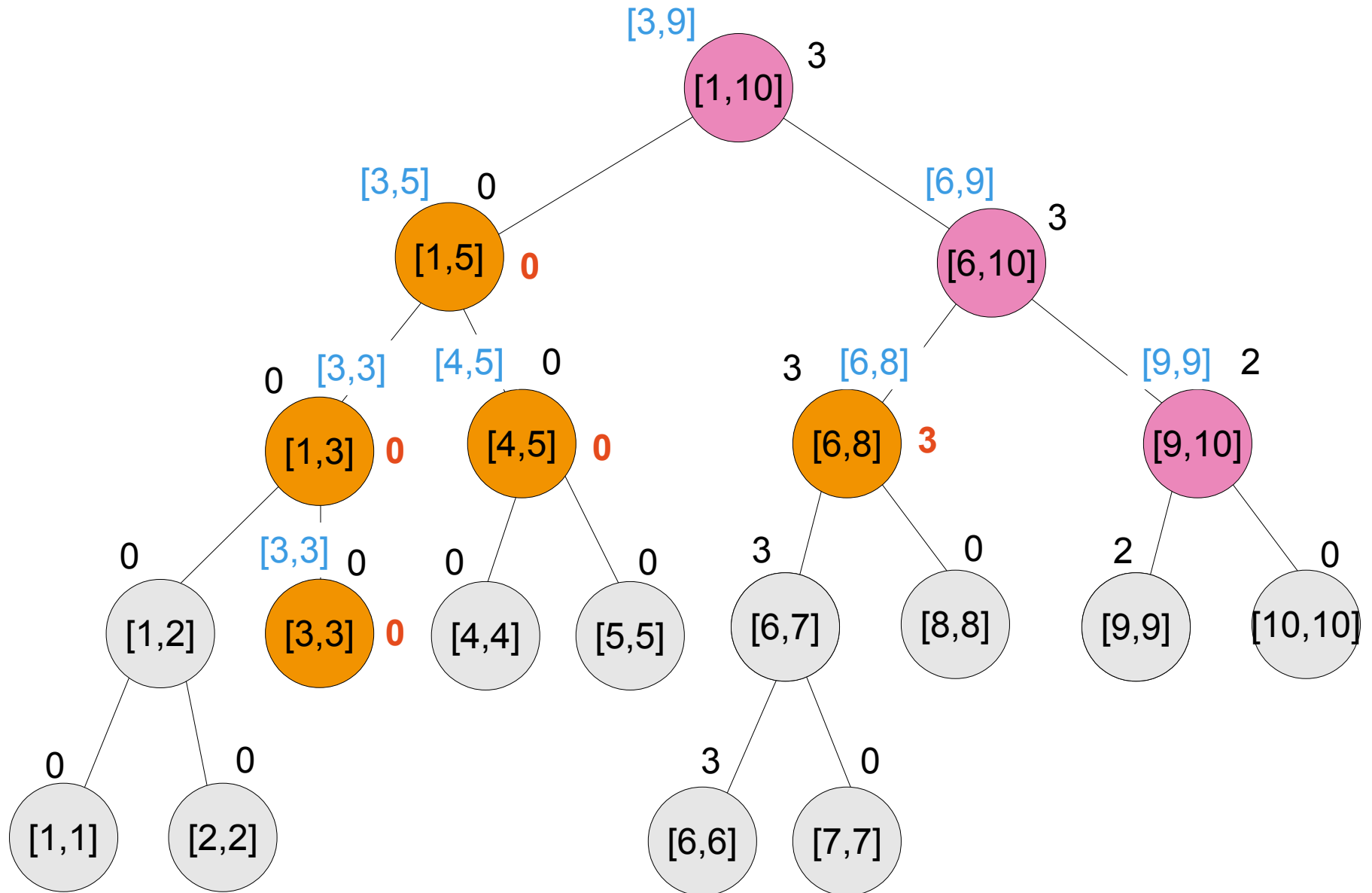
Query [3, 9]

# Interval Trees



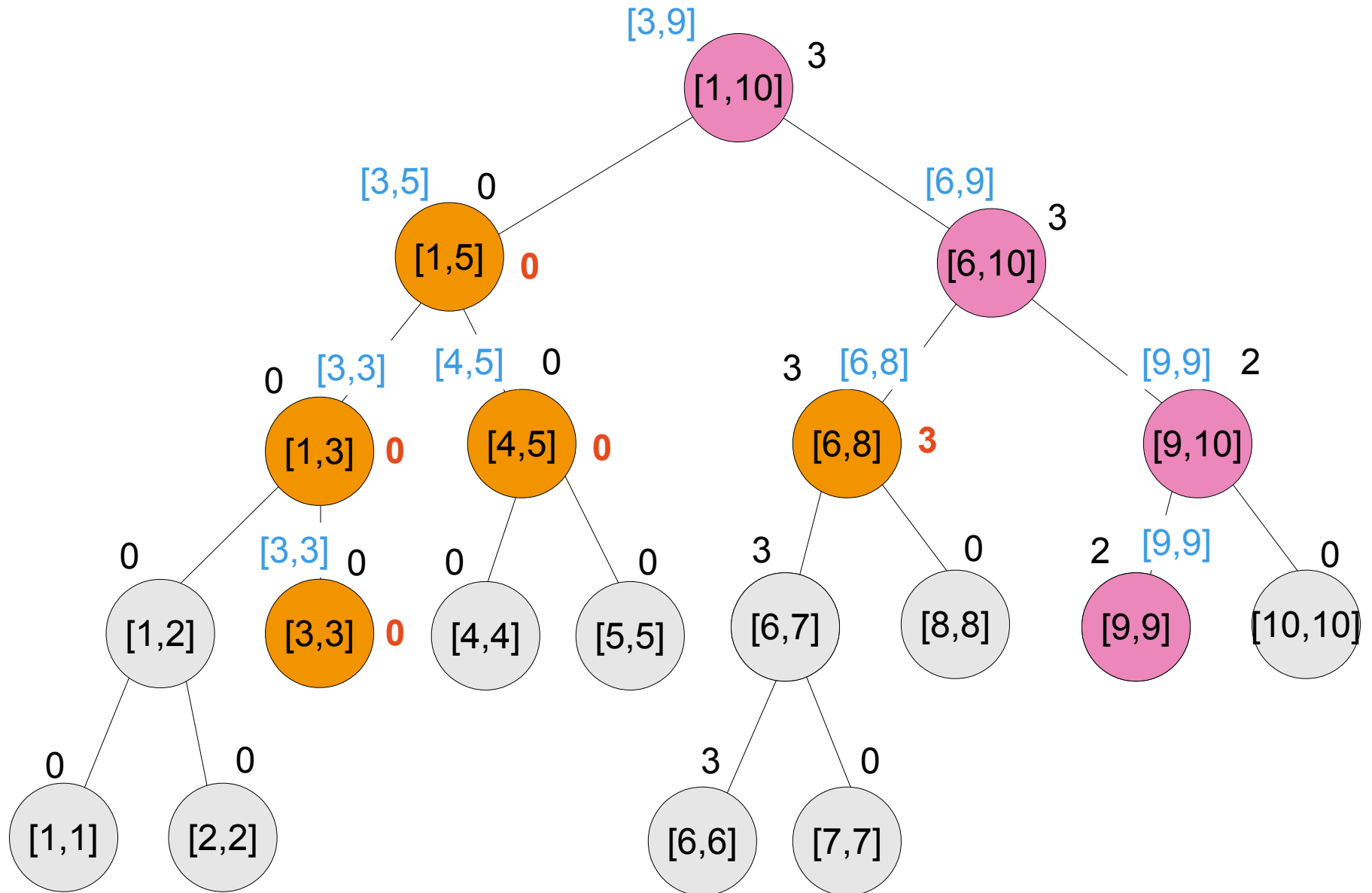
Query [3, 9]

# Interval Trees



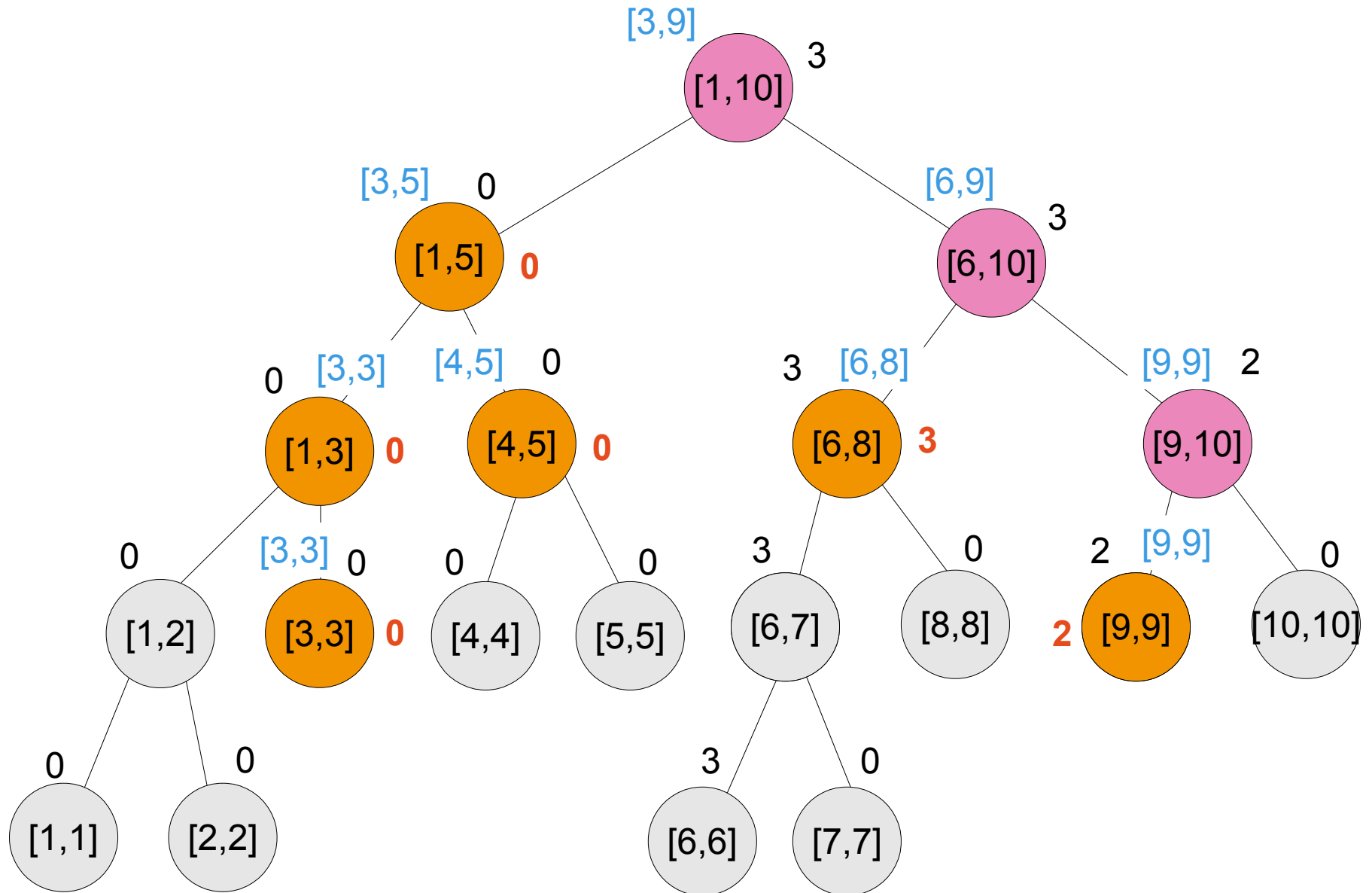
Query [3, 9]

# Interval Trees



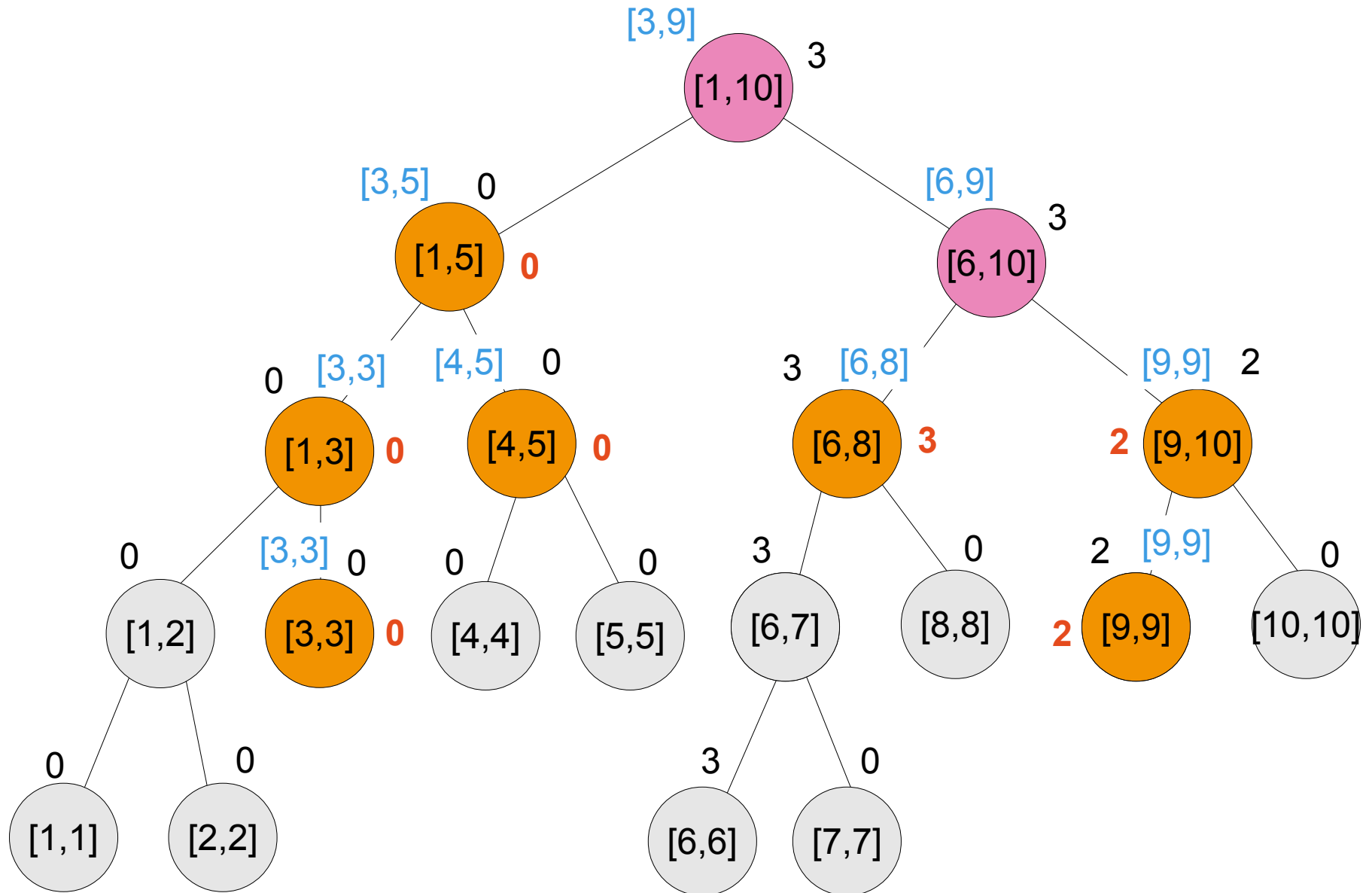
Query [3, 9]

# Interval Trees



Query [3, 9]

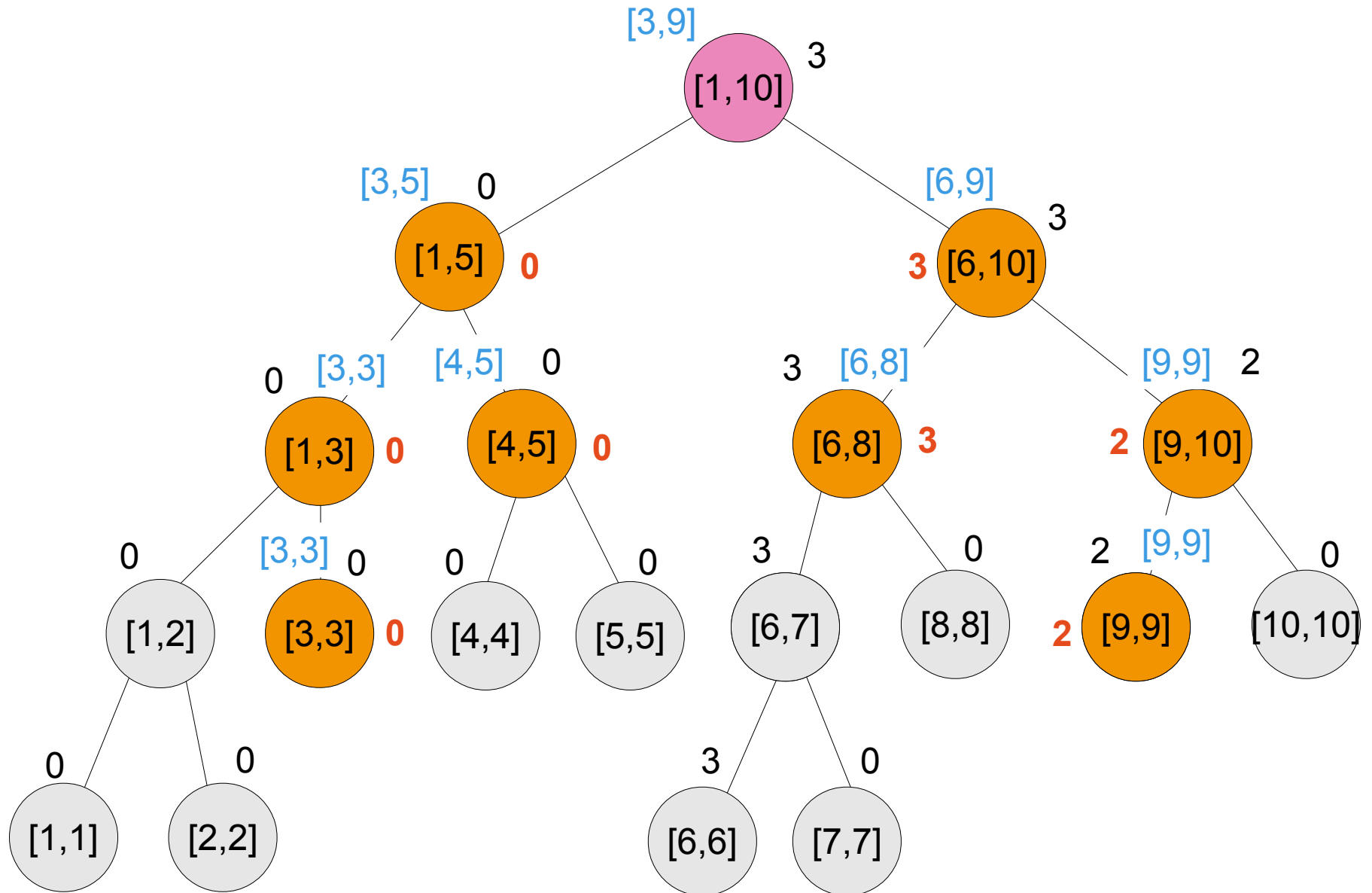
# Interval Trees





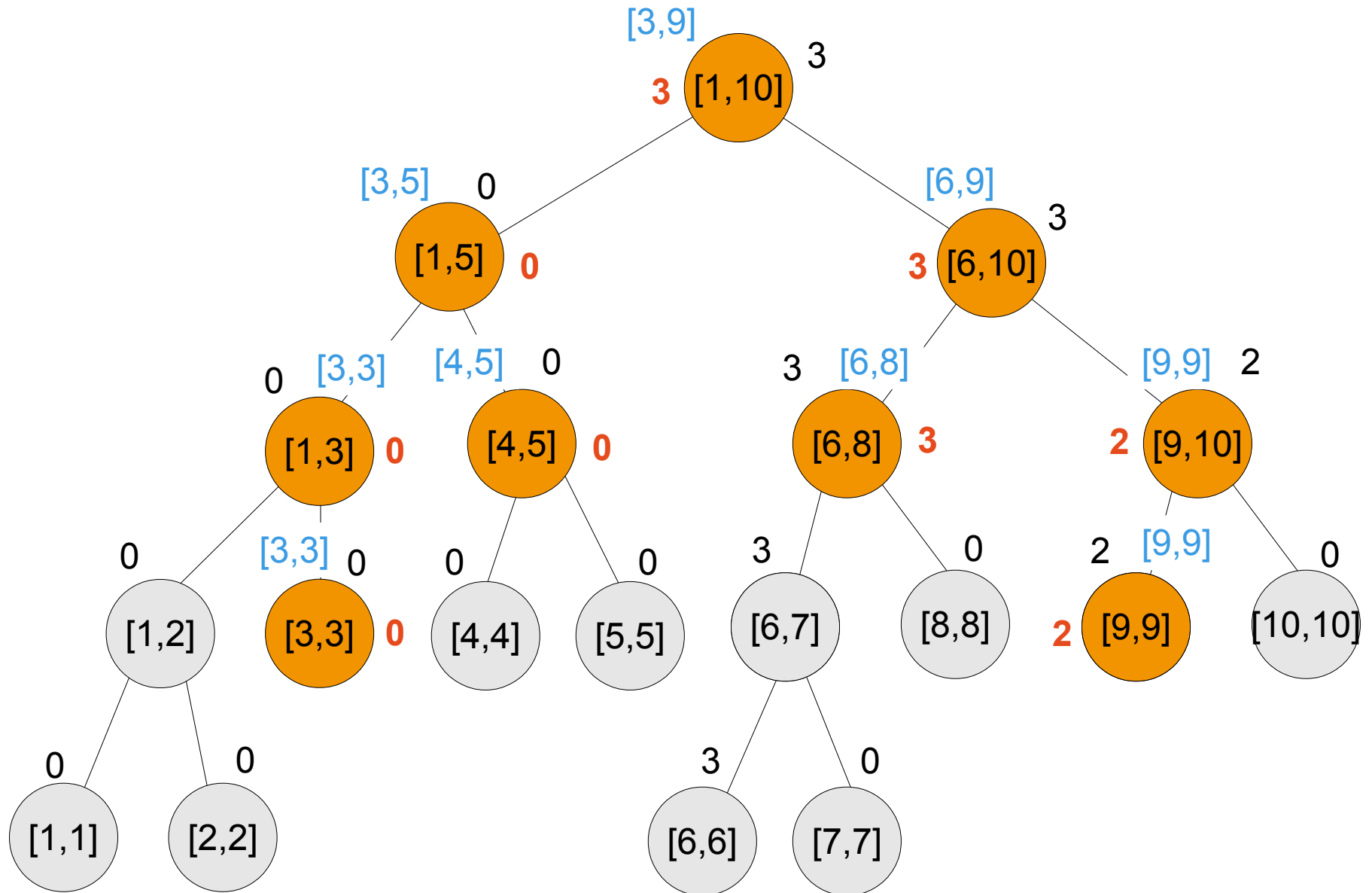
Query [3, 9]

# Interval Trees

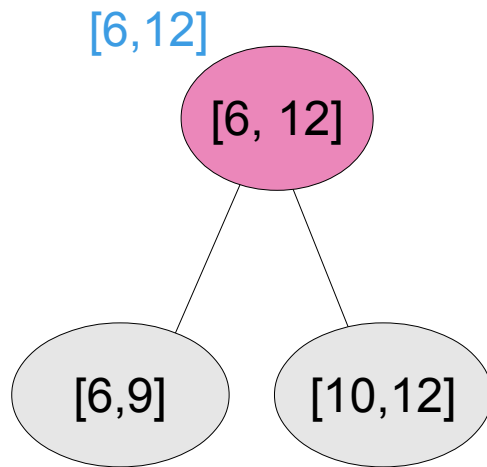


Query [3, 9]

# Interval Trees

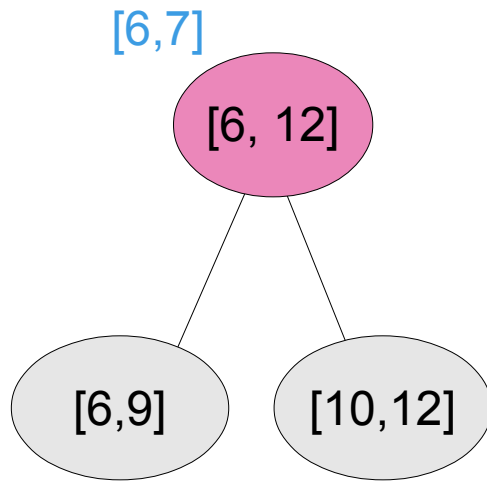


# Περίπτώσεις ερωτημάτων (1η)

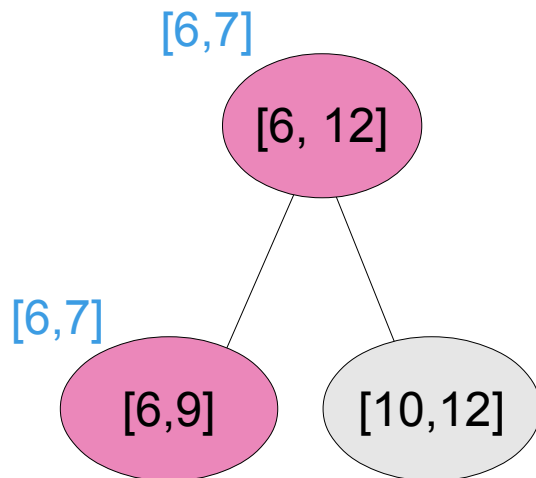


$QX = x$  και  $QY = y$   
επιστρέφουμε  $T[id]$

# Περίπτώσεις ερωτημάτων (2η)

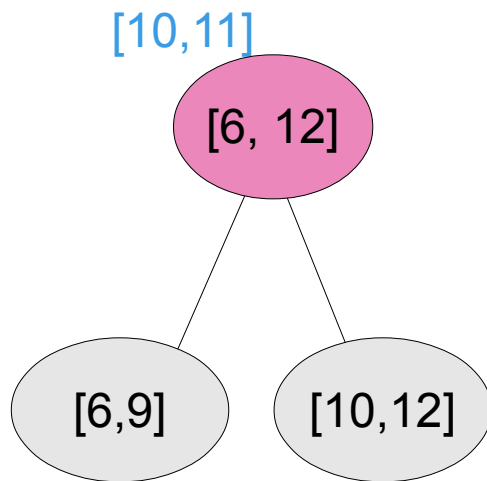


$QY \leq mid$   
μας ενδιαφέρει μόνο το αριστερό κλαδί

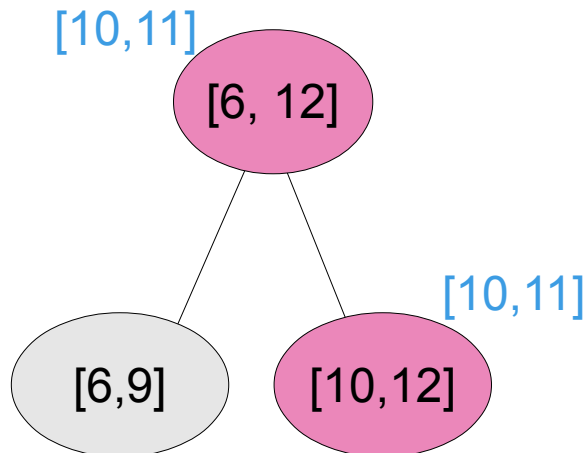


Επιστρέφουμε το αποτέλεσμα του αριστερού κλαδιού  
στο query με το ίδιο διάστημα

# Περιπτώσεις ερωτημάτων (3η)

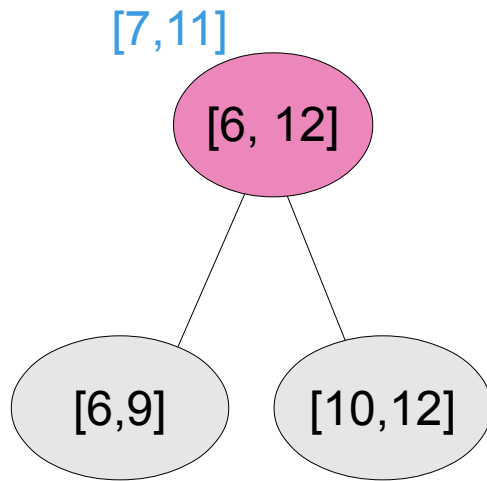


$QX \geq \text{mid} + 1$   
μας ενδιαφέρει μόνο το δεξιό κλαδί

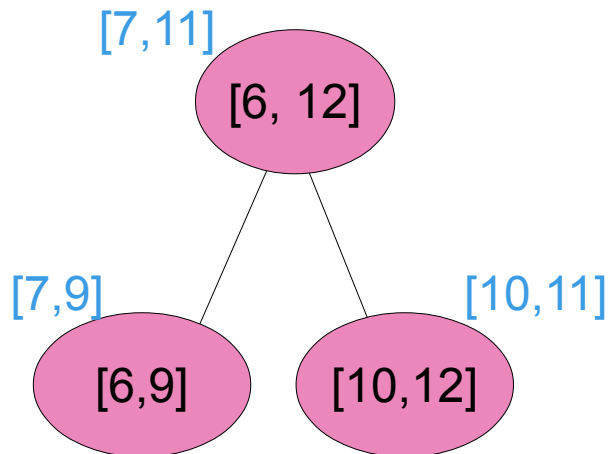


Επιστρέφουμε το αποτέλεσμα του δεξιού κλαδιού  
στο query με το ίδιο διάστημα

# Περιπτώσεις ερωτημάτων (4η)



Όταν δεν ισχύει καμία από τις προηγούμενες συνθήκες μας ενδιαφέρουν και τα δύο κλαδιά.



Σπάμε το διάστημα ερωτήματος σε δύο ανεξάρτητα Διαστήματα τα οποία στέλνουμε στα δύο παιδιά.

# Interval Trees

```
int query(int QX, int QY, int x, int y, int id) {  
    if ( x == QX && y == QY ) return T[id];  
  
    mid = (x + y)/2;  
    if ( QY <= mid ) {  
        return query(QX, QY, x, mid, 2*id);  
    }  
    else if ( QX > mid ) {  
        return query(QX, QY, mid+1, y, 2*id+1);  
    }  
    else {  
        return max(query(QX, mid, x, mid, 2*id),  
                   query(mid+1, QY, mid+1, y, 2*id + 1));  
    }  
}
```

```
query(QX, QY, 1, N, 1);
```