

Προχωρημένες Δομές Δεδομένων

Απρίλιος 2011

Γιάννης Χατζημίχος
feedward@gmail.com

Προκατασκευαστικό Camp 23ου 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 δενρική 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 ως αναπαράσταση λεξικών.

Tries: Πράξεις

- Προσθήκη συμβολοσειράς
- Αφαίρεση συμβολοσειράς
- Έλεγχος για ύπαρξη συμβολοσειράς
- Αναζήτηση/καταμέτριση συμβολοσειρών με το ίδιο πρόθεμα
- Αλφαριθμητική ταξινόμηση συμβολοσειρών
 - Εύρεση κ-οστης λέξης
- ...

Tries: Κατασκευή

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

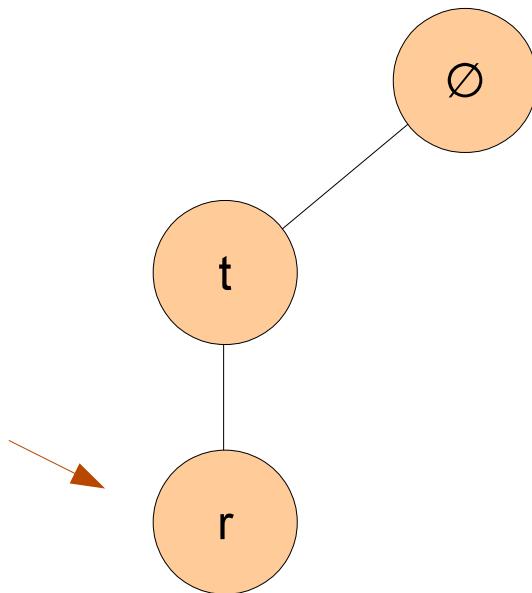
Tries: Παράδειγμα κατασκευής



Tries: Παράδειγμα κατασκευής

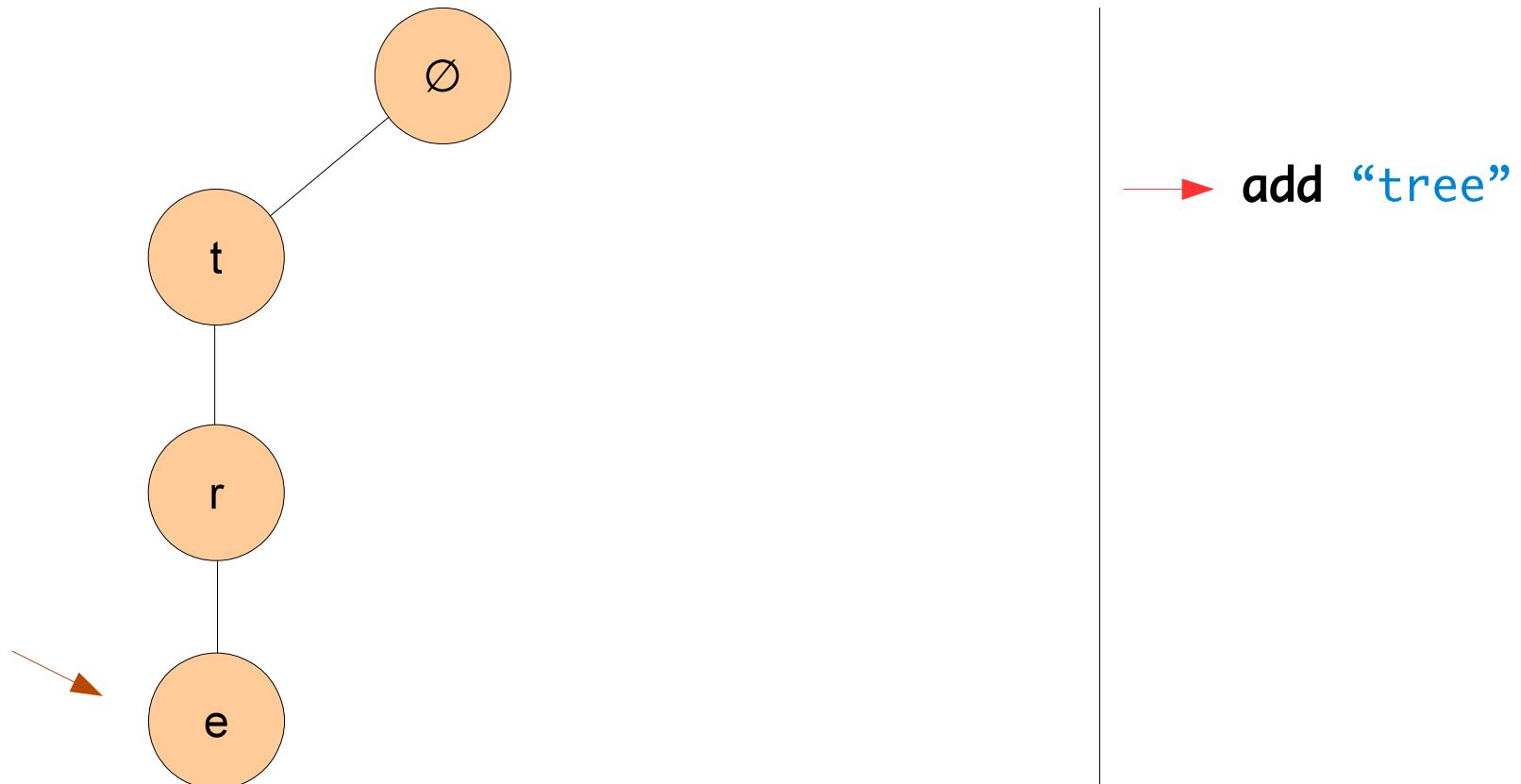


Tries: Παράδειγμα κατασκευής

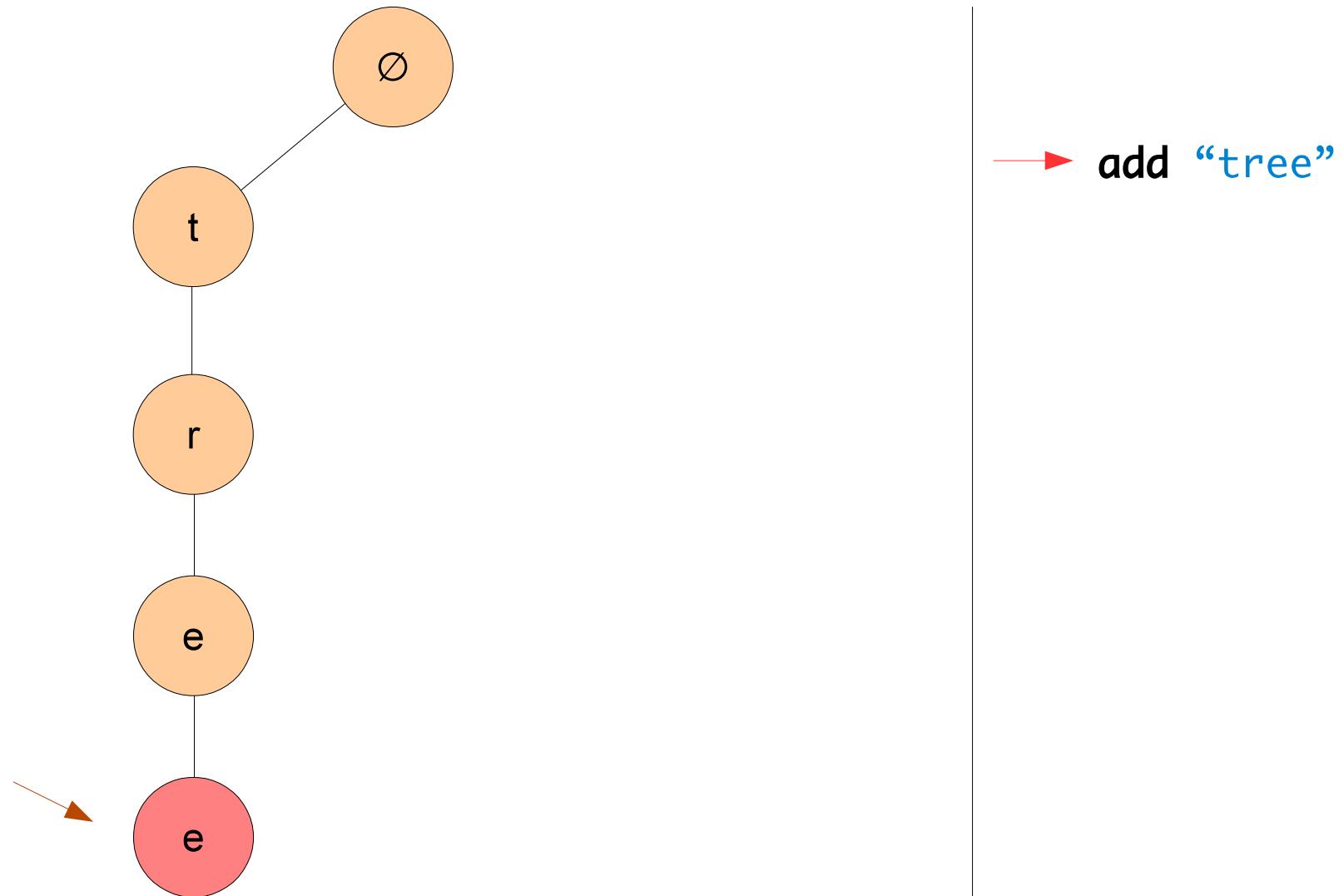


→ add “tree”

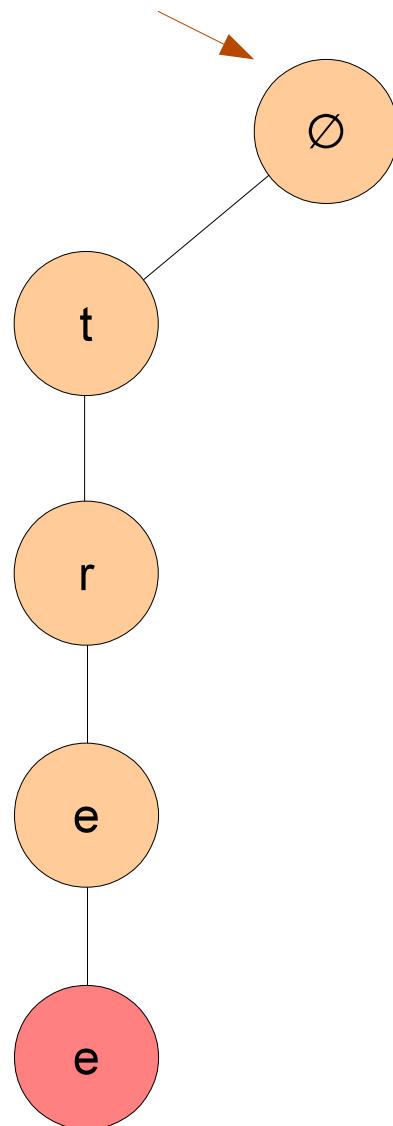
Tries: Παράδειγμα κατασκευής



Tries: Παράδειγμα κατασκευής

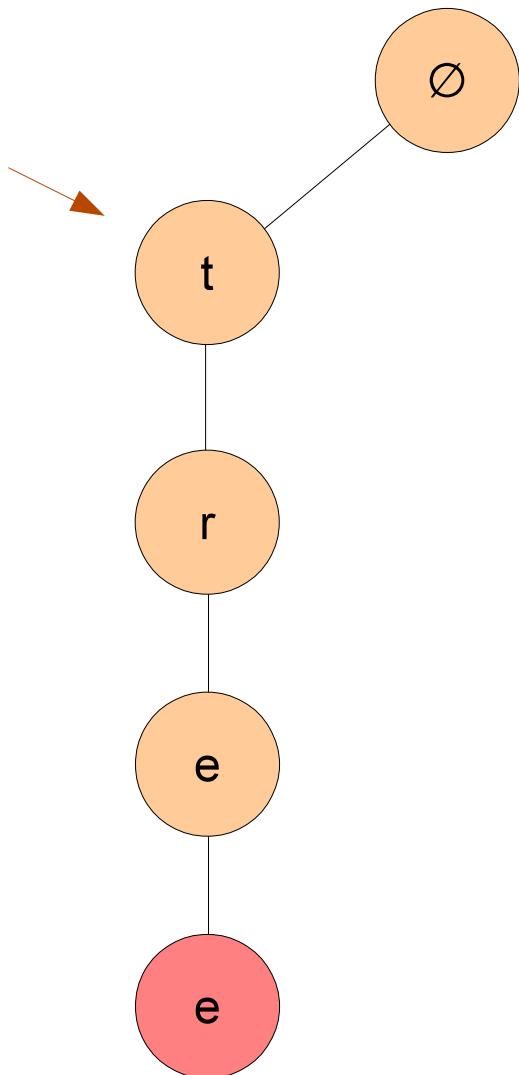


Tries: Παράδειγμα κατασκευής



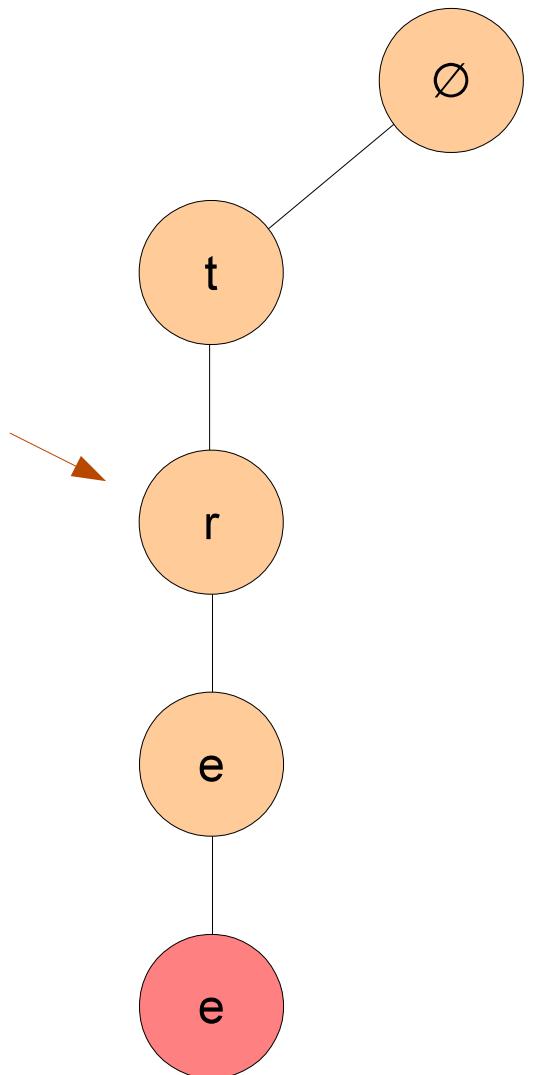
add “tree”
→ add “trie”

Tries: Παράδειγμα κατασκευής



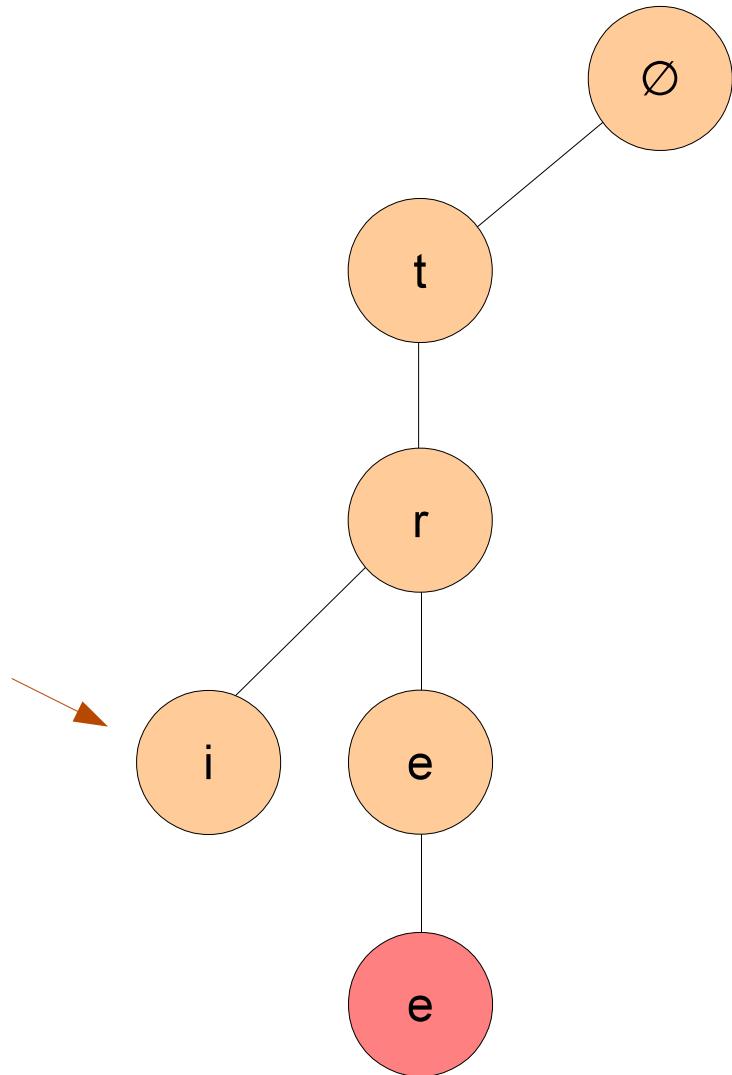
add “tree”
add “trie”

Tries: Παράδειγμα κατασκευής



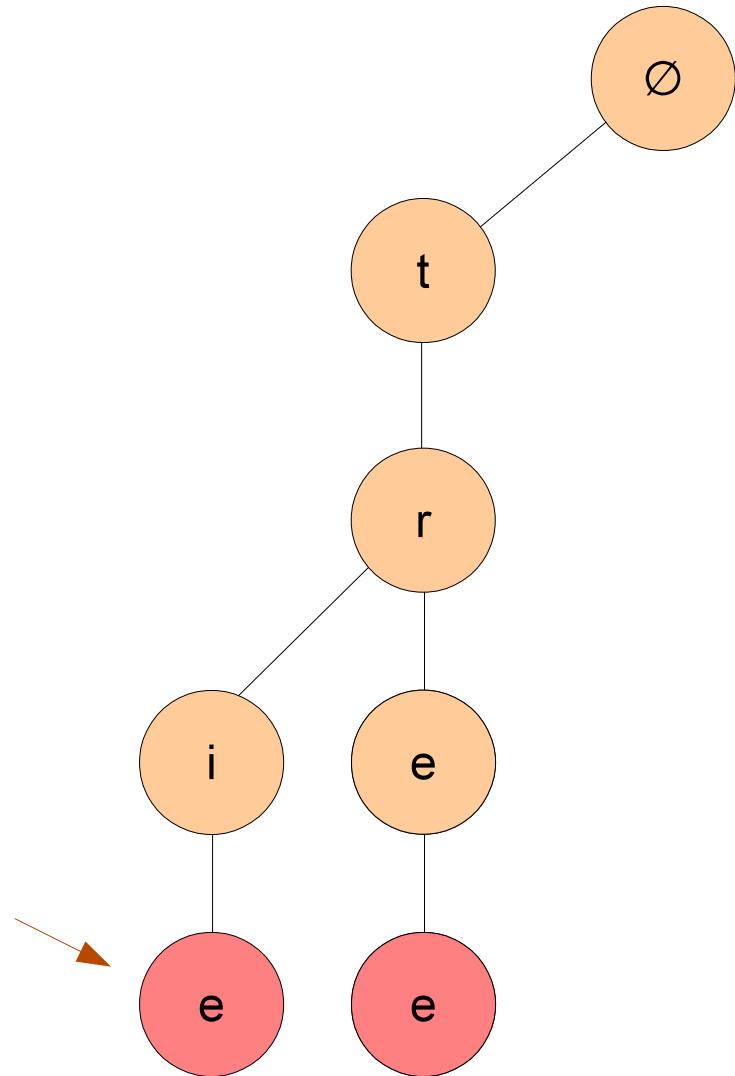
add “tree”
→ add “trie”

Tries: Παράδειγμα κατασκευής



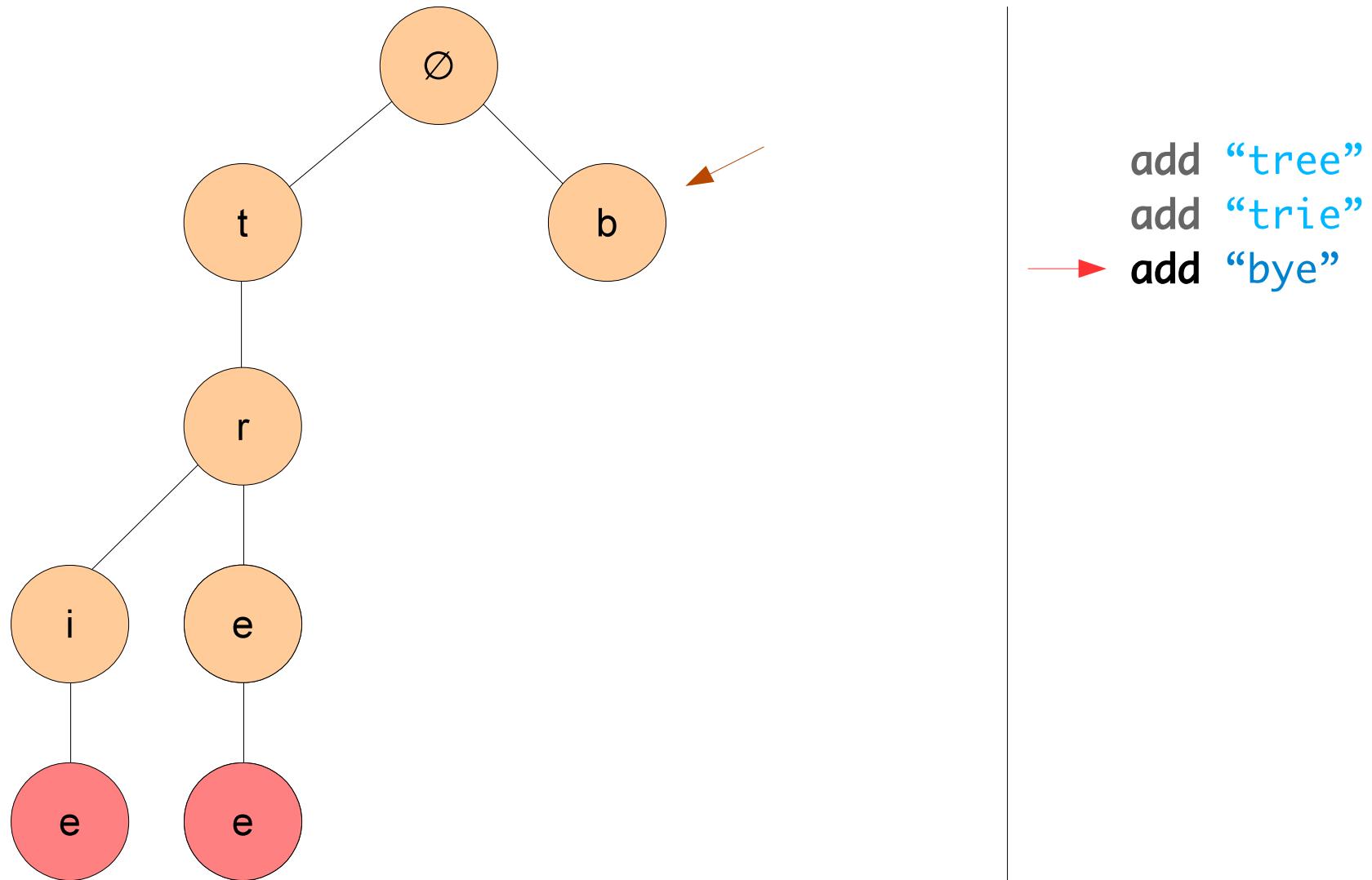
add “tree”
→ add “trie”

Tries: Παράδειγμα κατασκευής

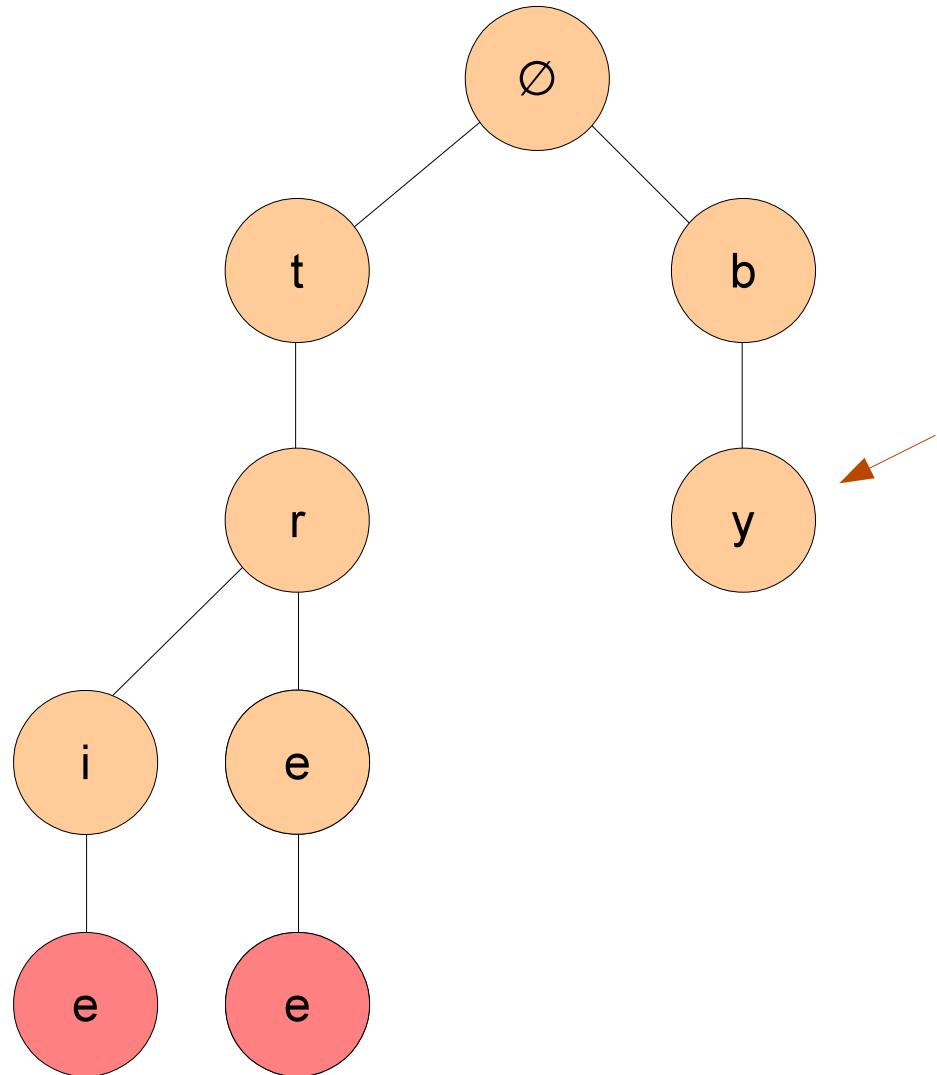


add “tree”
→ add “trie”

Tries: Παράδειγμα κατασκευής

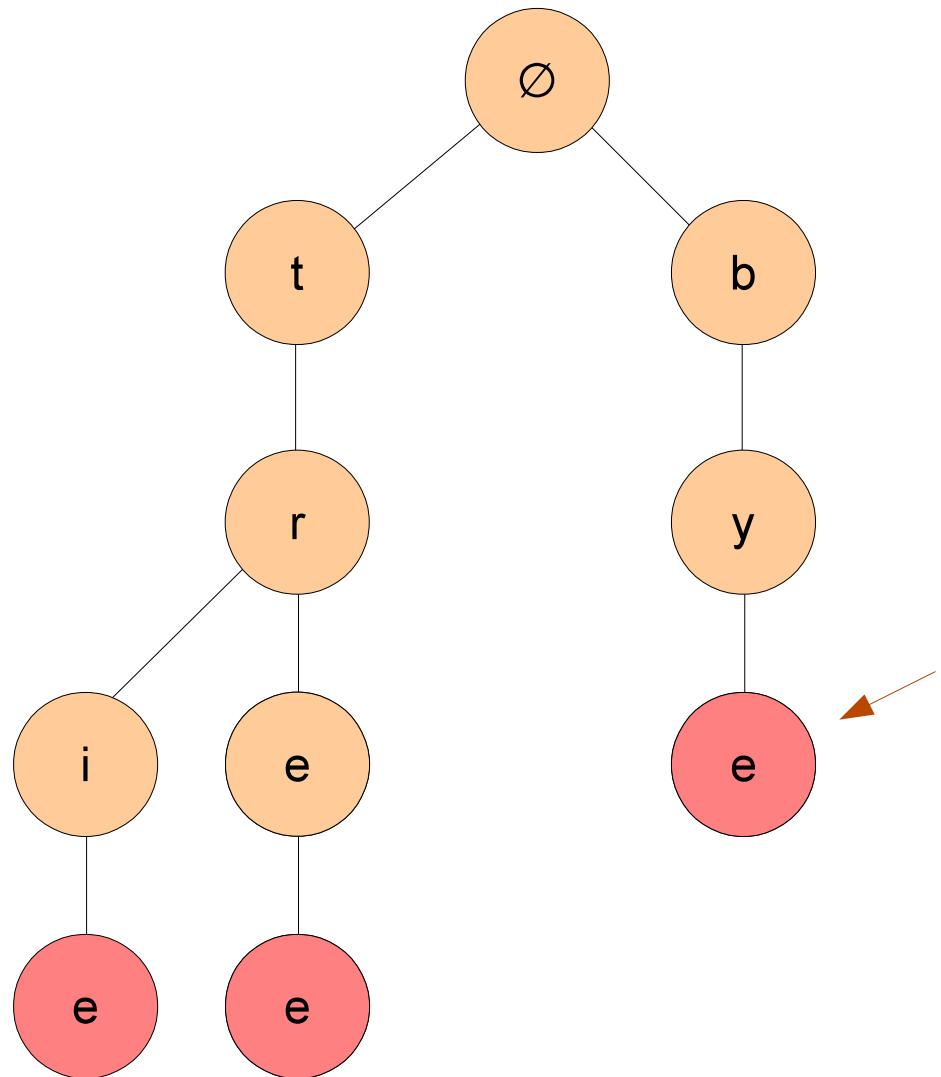


Tries: Παράδειγμα κατασκευής



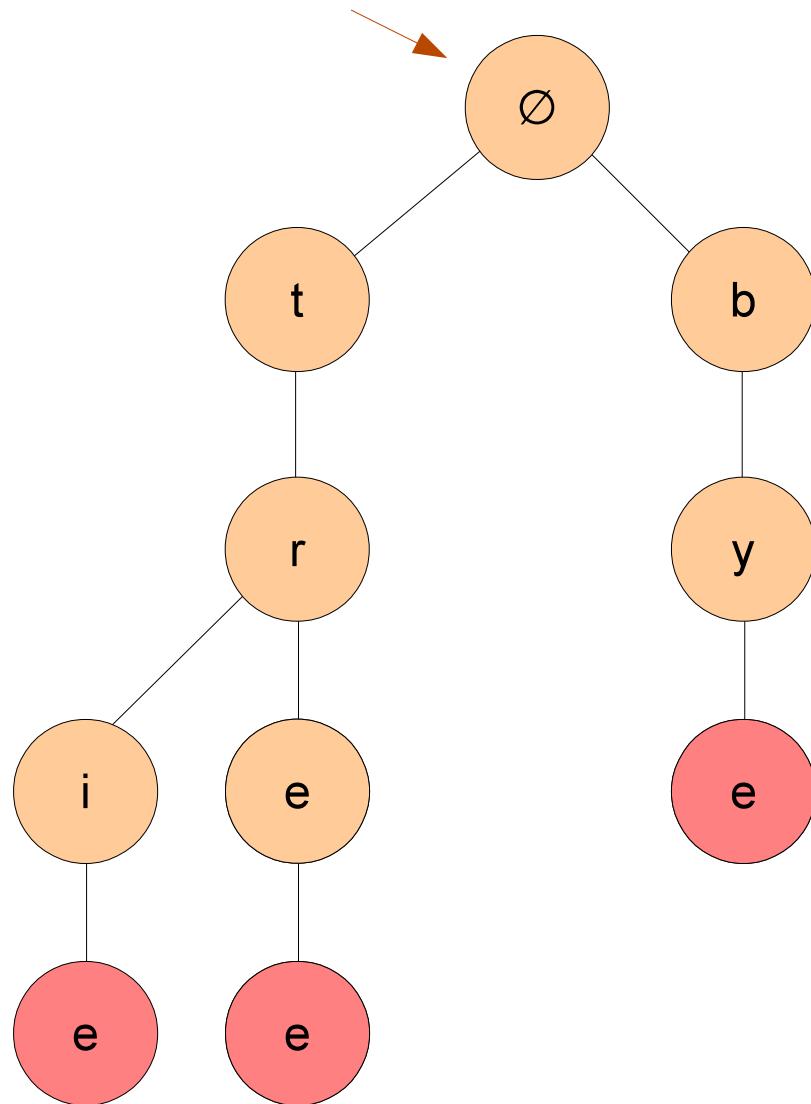
add “tree”
add “trie”
→ add “bye”

Tries: Παράδειγμα κατασκευής



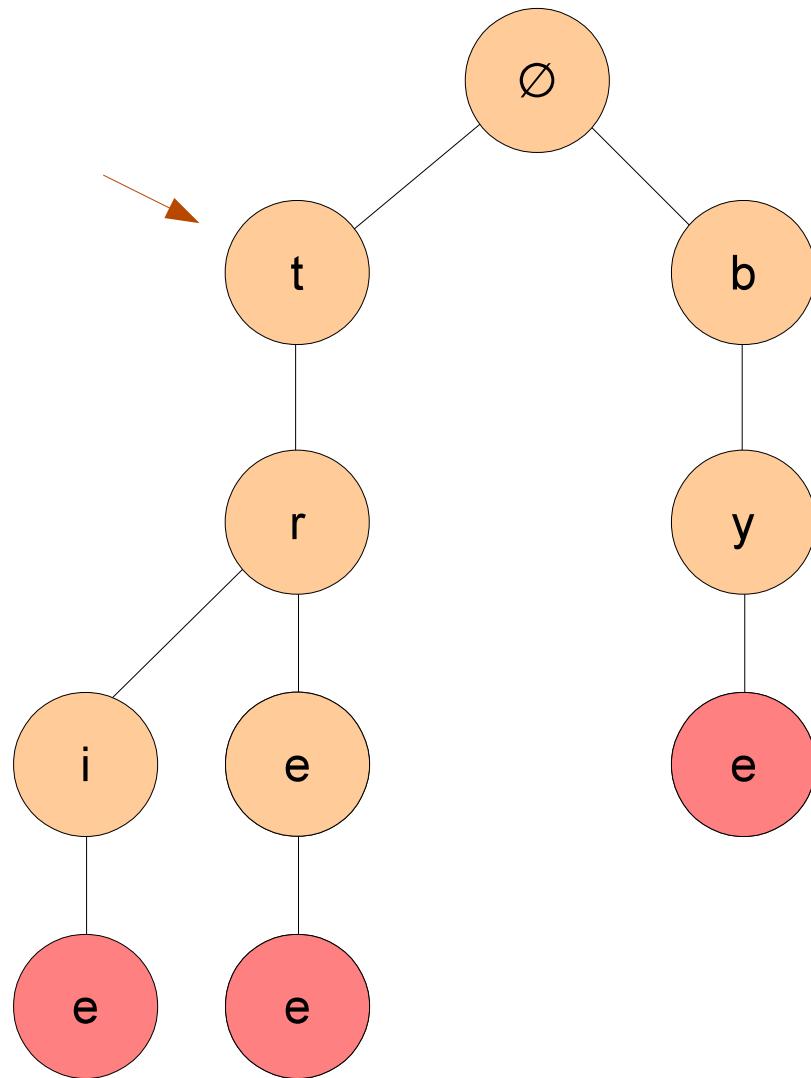
add “tree”
add “trie”
→ add “bye”

Tries: Παράδειγμα κατασκευής



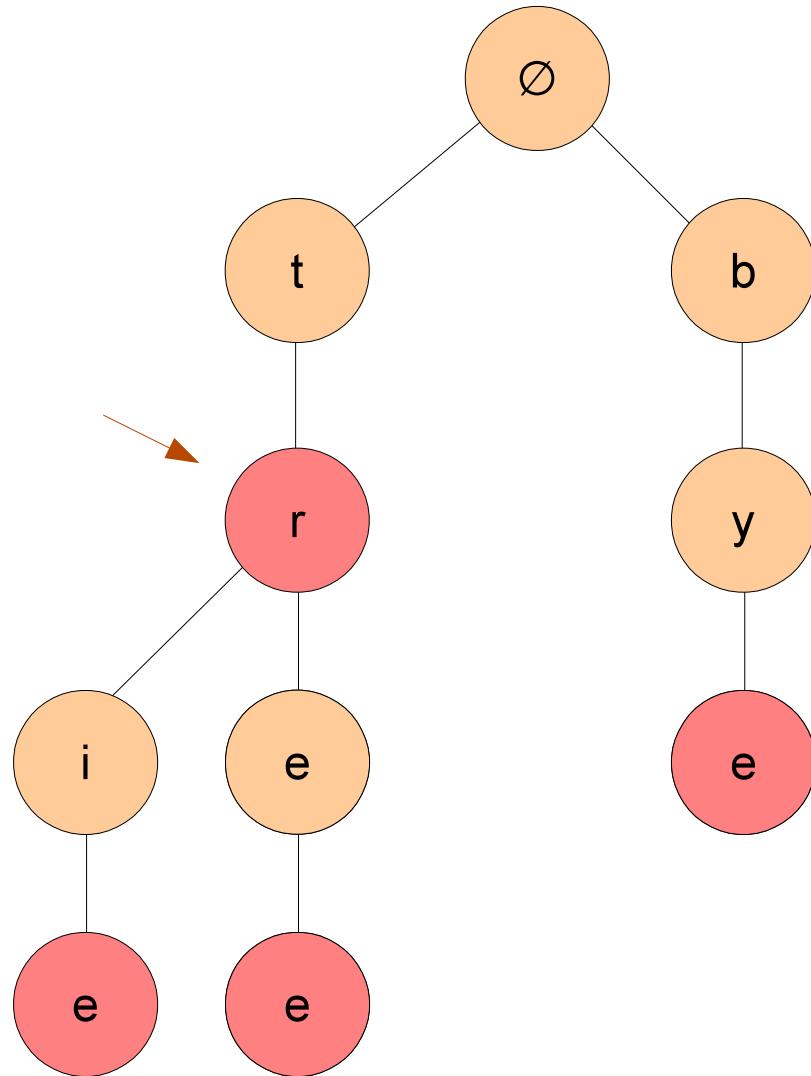
add “tree”
add “trie”
add “bye”
→ add “tr”

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

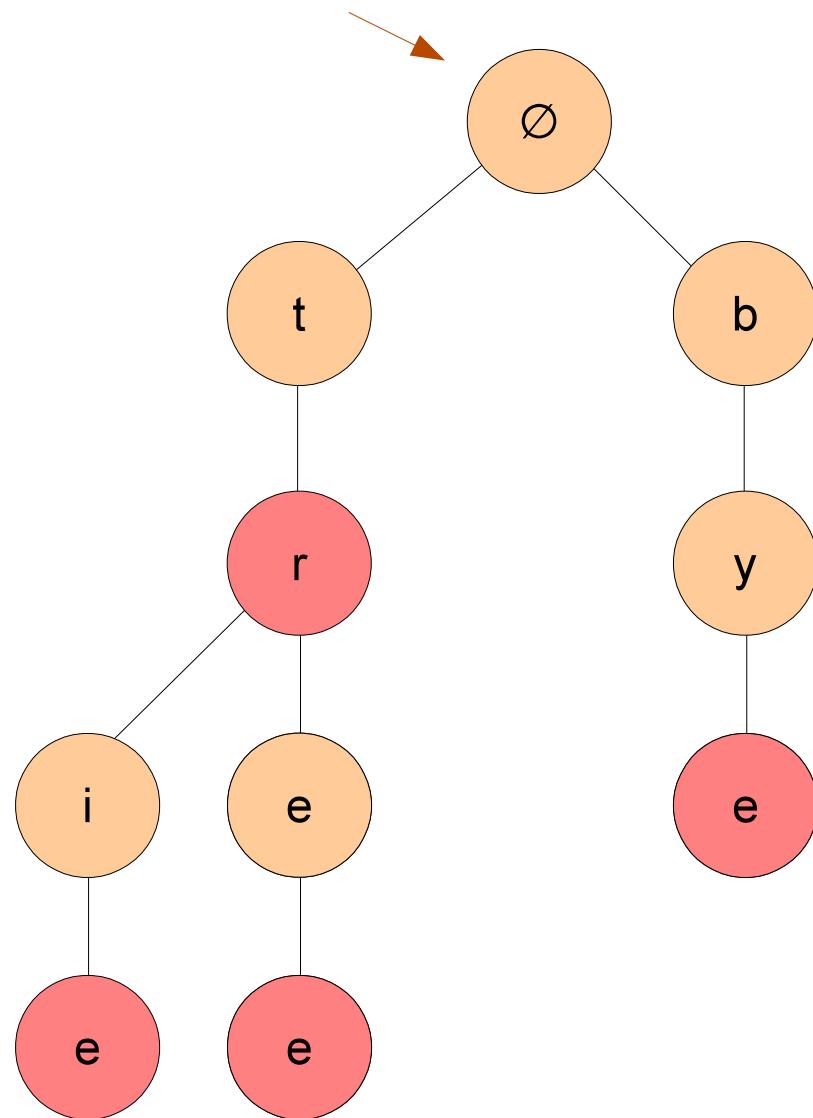
```
int add(char *word, int length) {
    int i, nextNode, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα */

    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 */

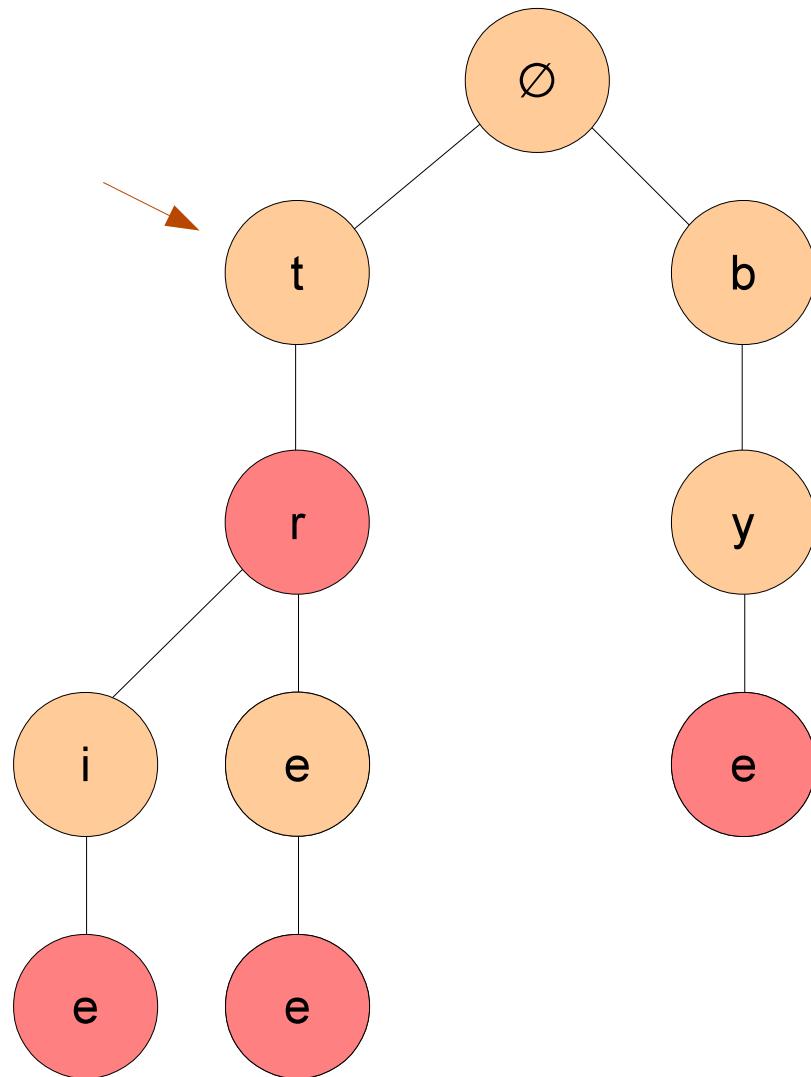
            trieNodeCount++;
            Trie[curNode].children[ word[i] - 'a' ] = trieNodeCount;
            curNode = trieNodeCount;
        }
        else {
            curNode = nextNode;
        }
    }
    Trie[curNode].isWord = 1;
}
```

Tries: Παραδείγματα



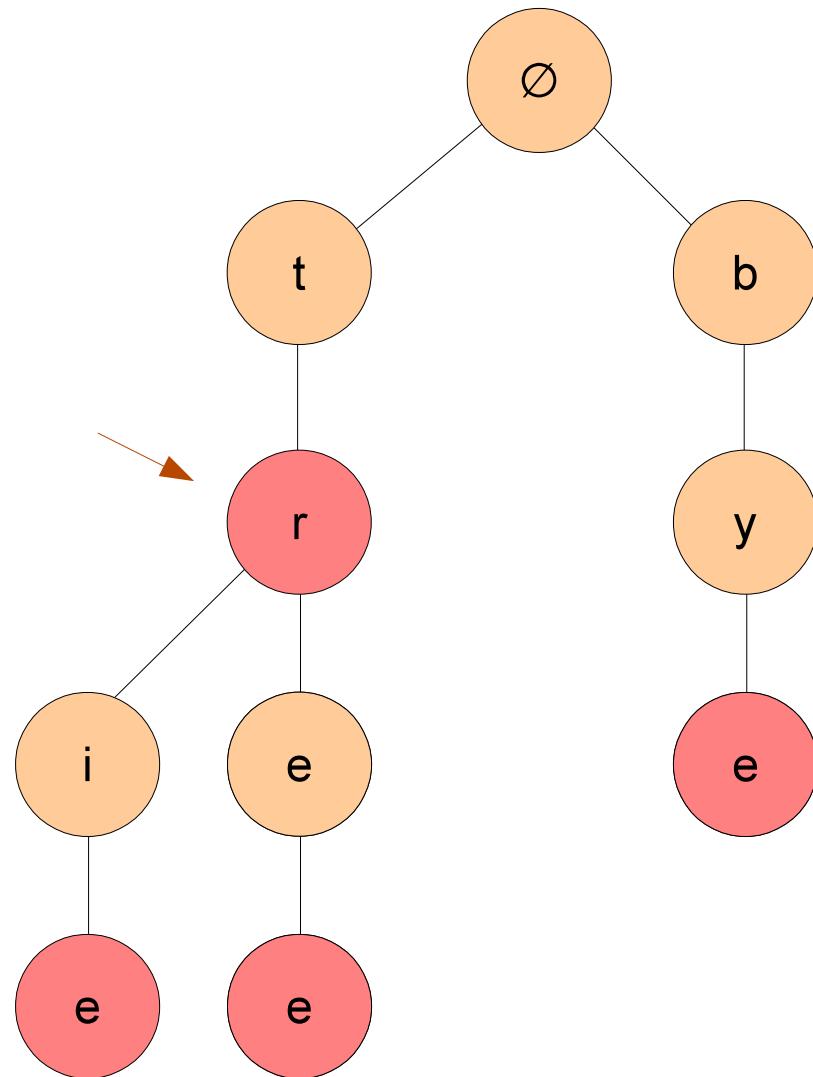
add “tree”
add “trie”
add “bye”
add “tr”
remove “trie”

Tries: Παραδείγματα



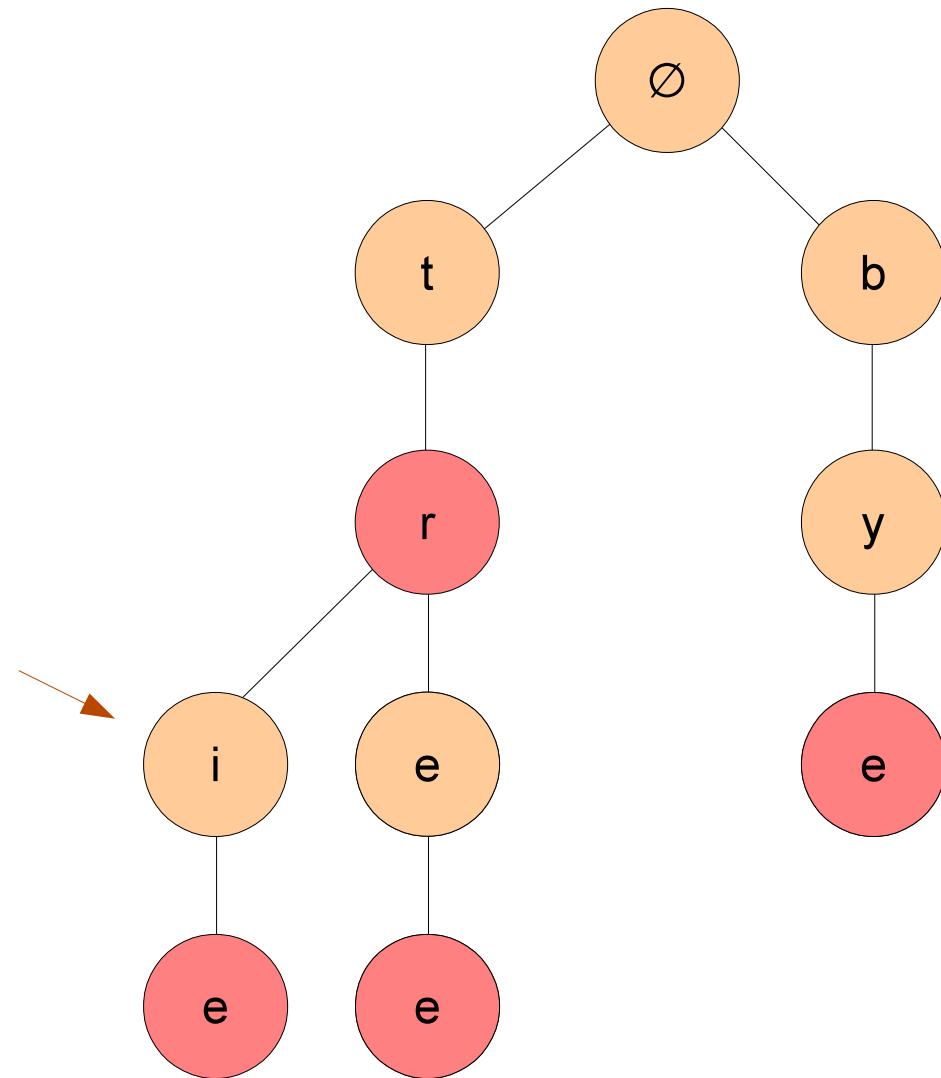
add "tree"
add "trie"
add "bye"
add "tr"
remove "trie"

Tries: Παραδείγματα



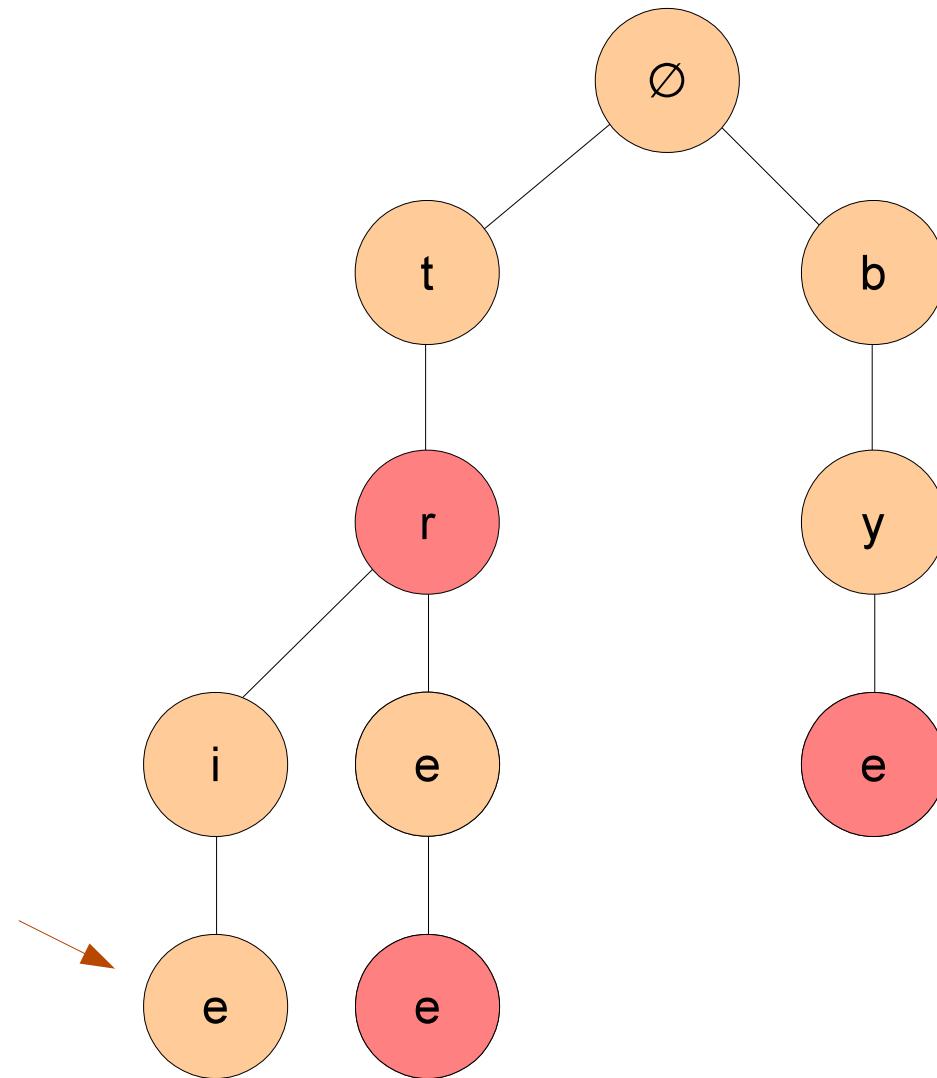
add "tree"
add "trie"
add "bye"
add "tr"
remove "trie"

Tries: Παραδείγματα



add “tree”
add “trie”
add “bye”
add “tr”
remove “trie”

Tries: Παραδείγματα



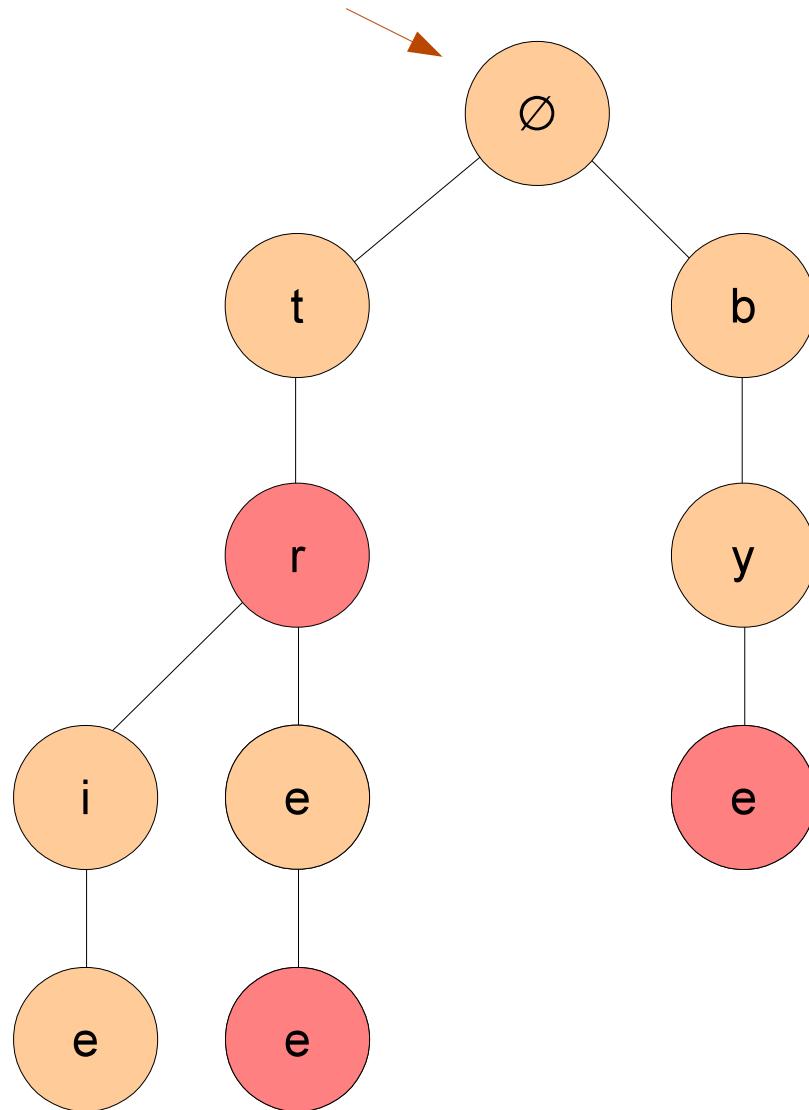
add “tree”
add “trie”
add “bye”
add “tr”
remove “trie”

αλλάζουμε το isWord σε 0

Tries: Υλοποίηση της remove

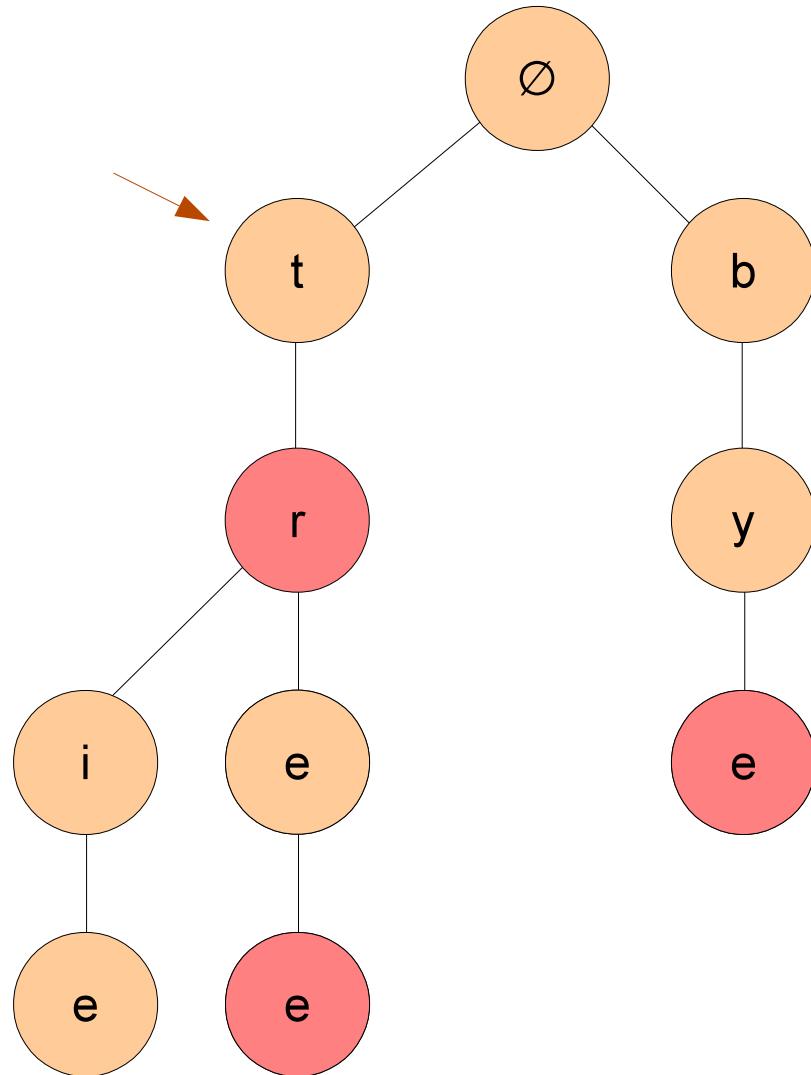
```
int remove(char word, int length) { /* ακολουθούμε το μονοπάτι μέχρι τον  
κόμβο του τελευταίου γράμματος και στη  
συνέχεια αλλάζουμε το isWord σε 0 */  
  
    int i, curNode = 1; /* τοποθετούμε το βέλος στην ρίζα του trie */  
    for (i=0; i<length; i++) {  
        curNode = Trie[curNode].children[word[i] - 'a'];  
  
        assert(curNode != 0); /* υποθέτουμε ότι η λέξη που θέλουμε να  
διαγράψουμε υπάρχει πάντα */  
    }  
    Trie[curNode].isWord = 0;  
}
```

Tries: Παραδείγματα



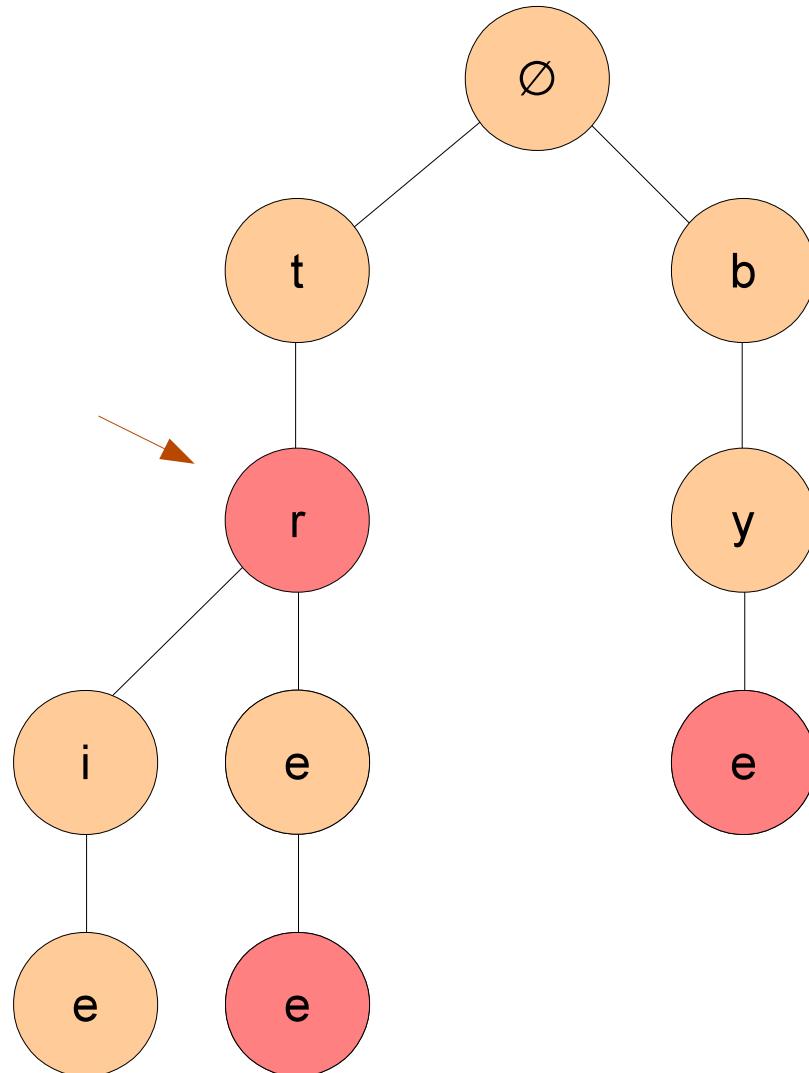
add "tree"
add "trie"
add "bye"
add "tr"
remove "trie"
check "tr"

Tries: Παραδείγματα



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

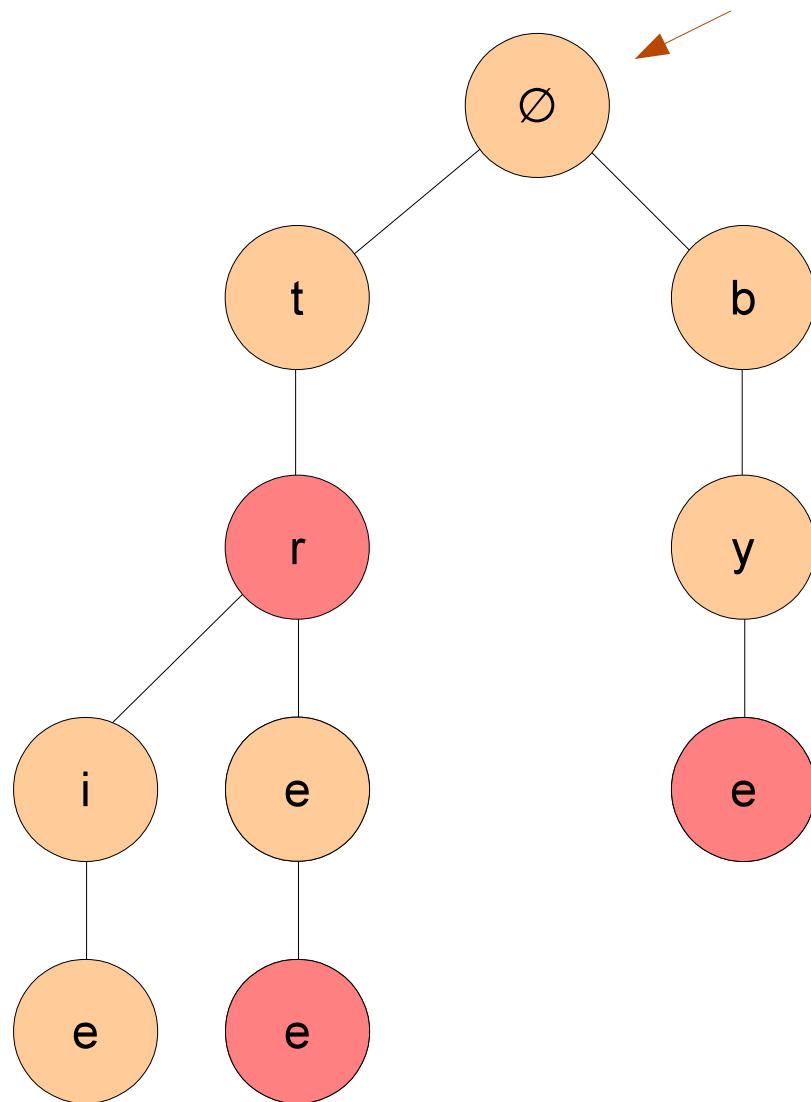
Tries: Παραδείγματα



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

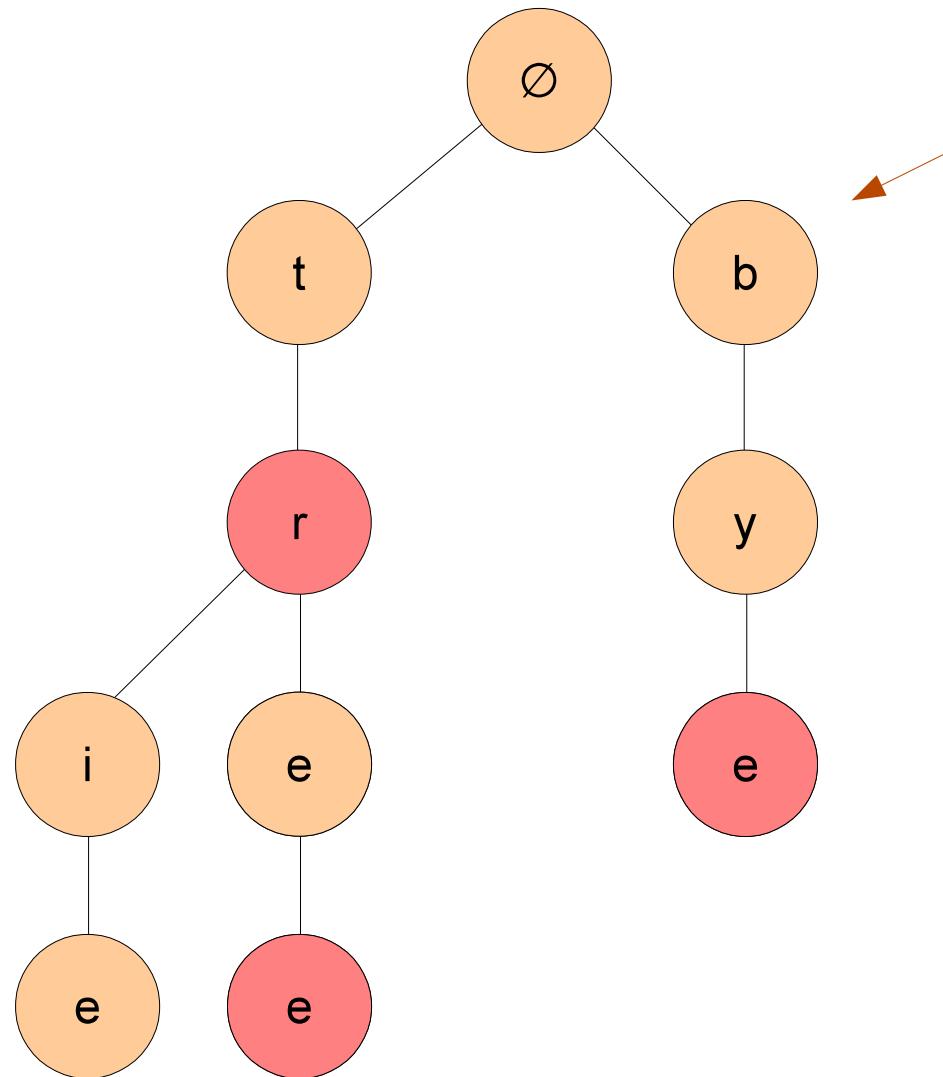
Φτάσαμε στο κόμβο του τελεύταιου γράμματος, ελέγχουμε αν το isWord αυτού του κόμβου είναι 1

Tries: Παραδείγματα



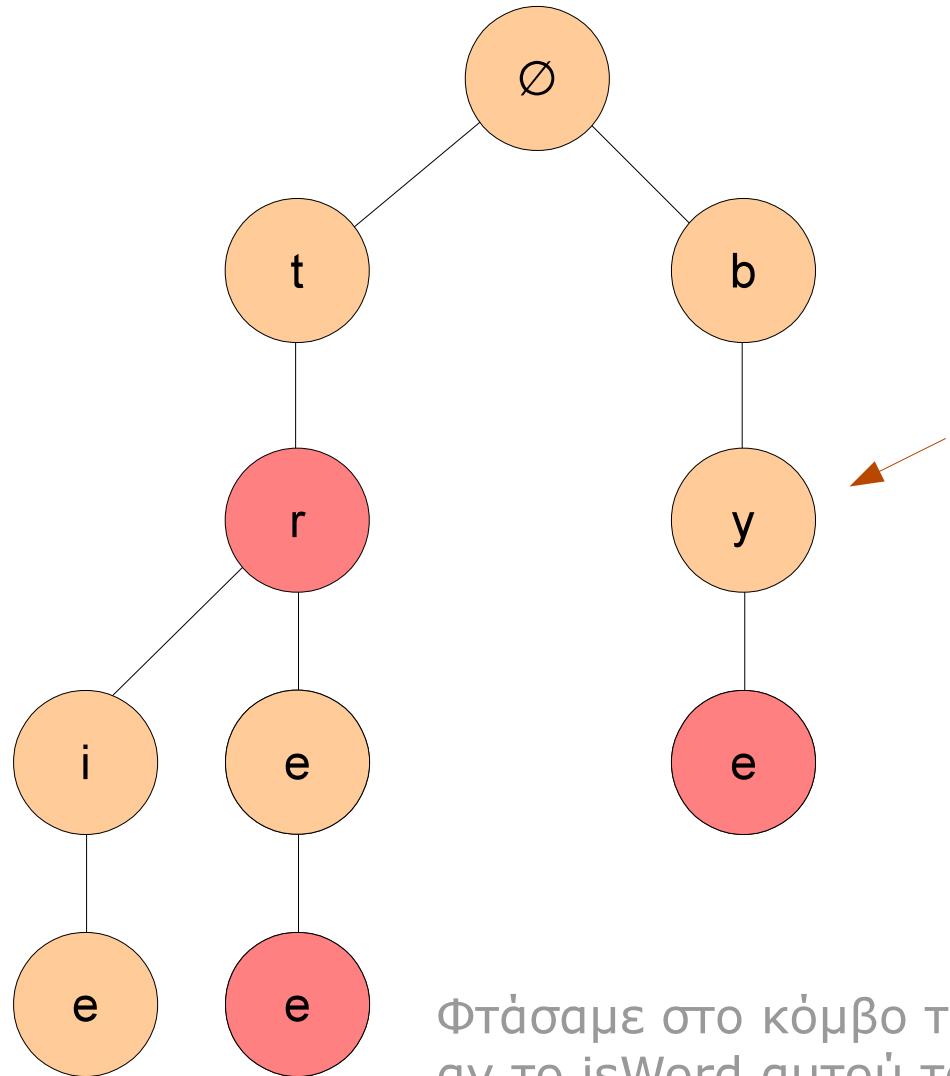
add "tree"
add "trie"
add "bye"
add "tr"
remove "trie"
check "tr" true
check "by"

Tries: Παραδείγματα



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

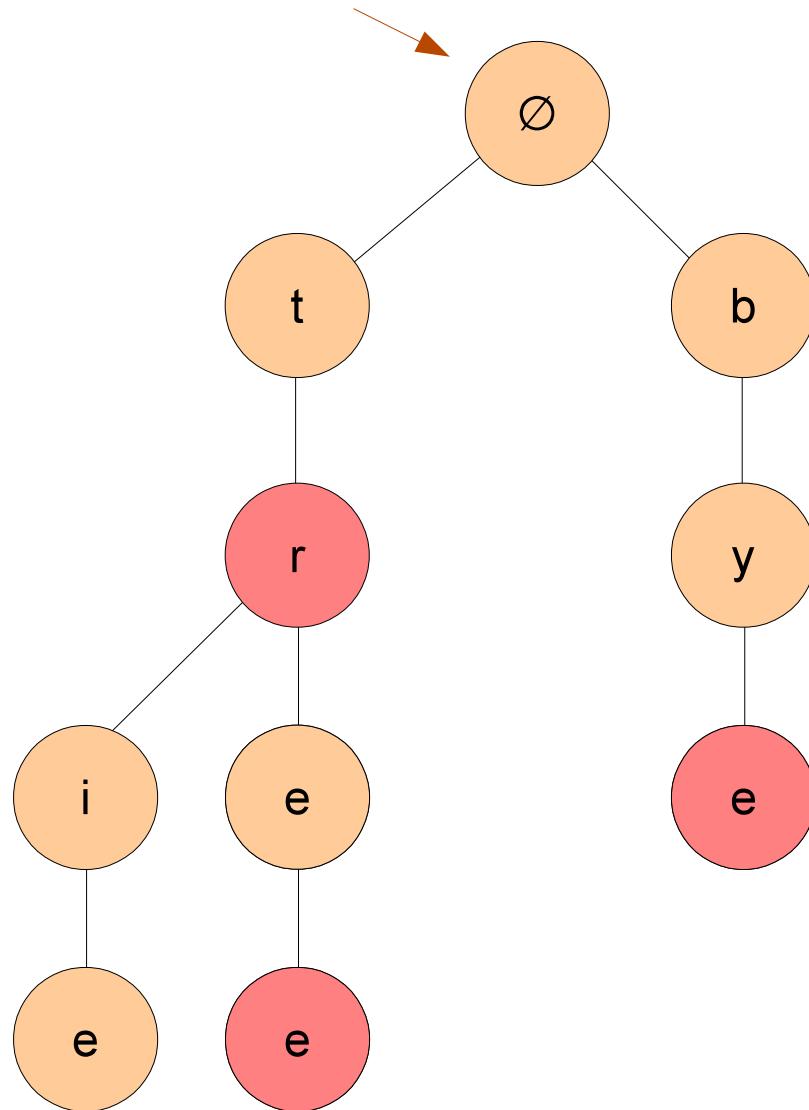
Tries: Παραδείγματα



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

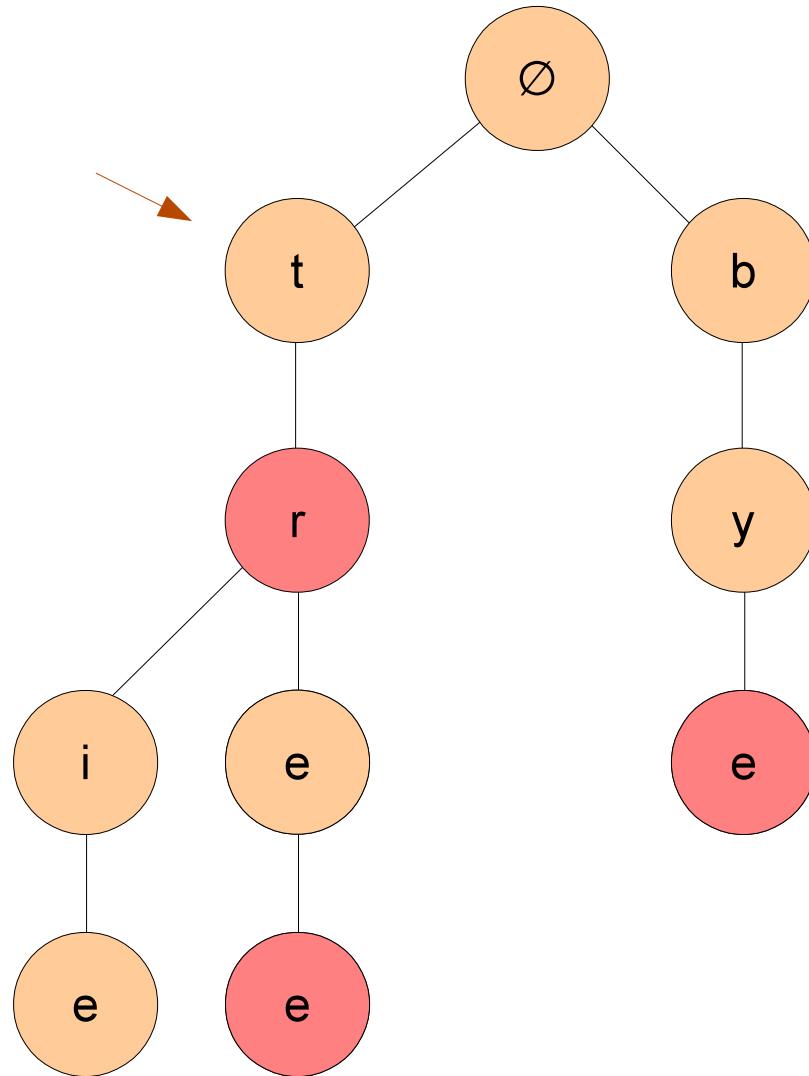
Φτάσαμε στο κόμβο του τελεύταιου γράμματος, ελέγχουμε αν το isWord αυτού του κόμβου είναι 1

Tries: Παραδείγματα



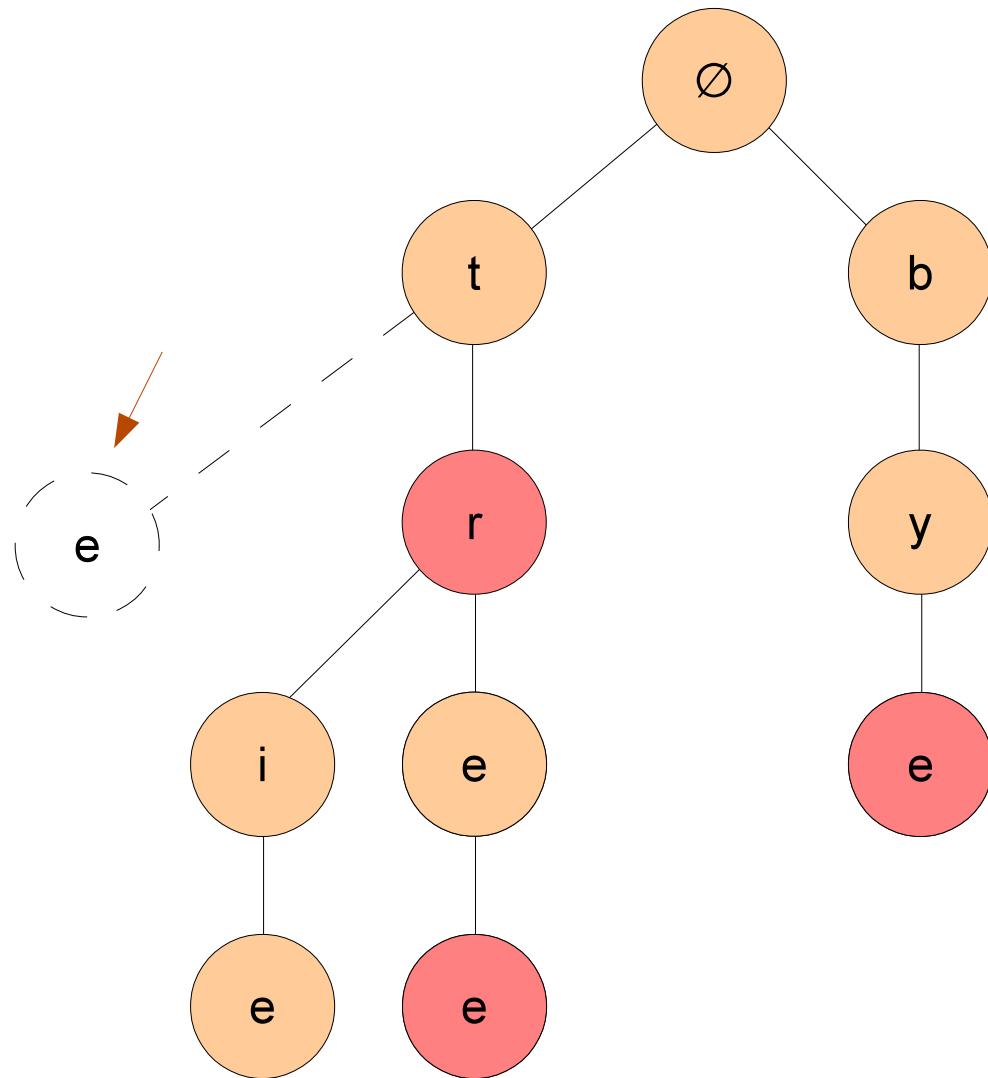
add "tree"
add "trie"
add "bye"
add "tr"
remove "trie"
check "tr" true
check "by" false
check "ten"

Tries: Παραδείγματα



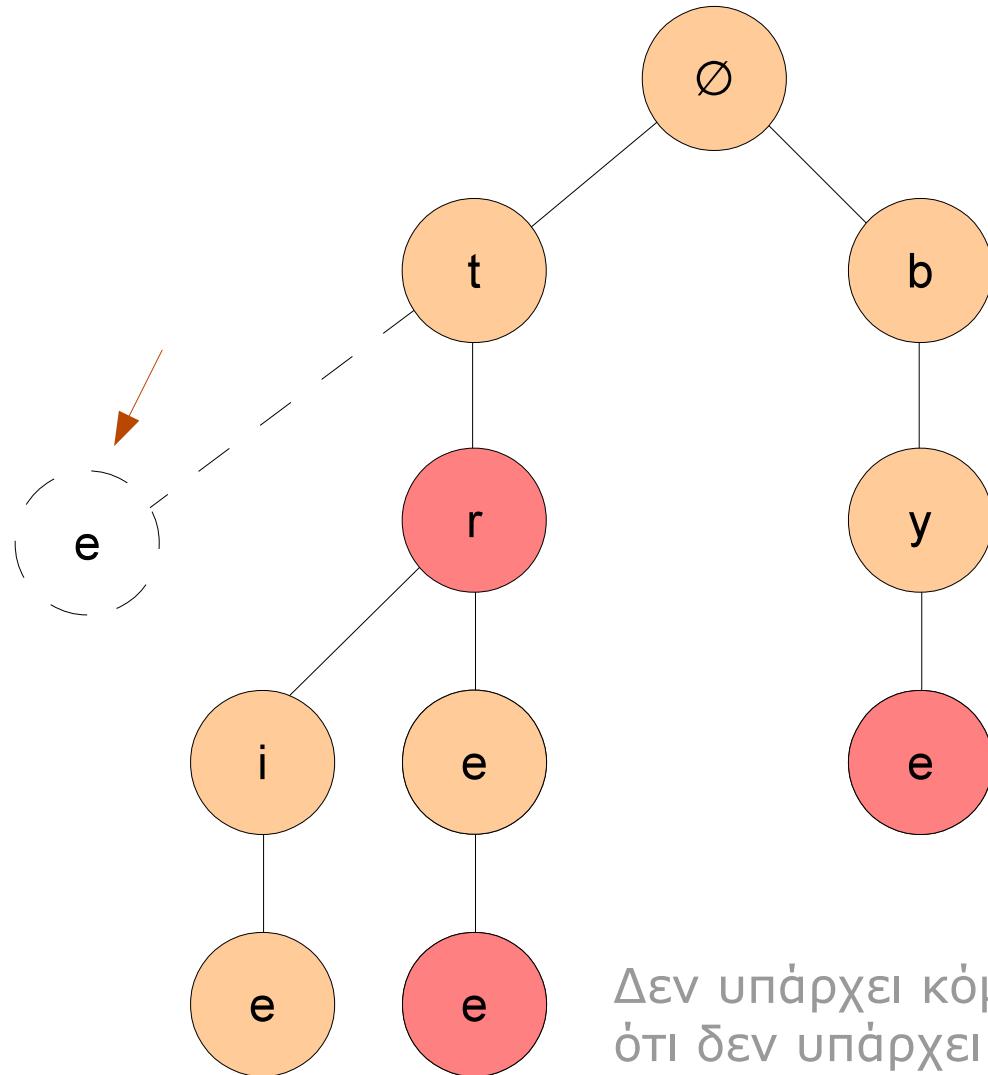
add "tree"
add "trie"
add "bye"
add "tr"
remove "trie"
check "tr" true
check "by" false
check "ten"

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 ωστέ να επιστρέψει 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
```

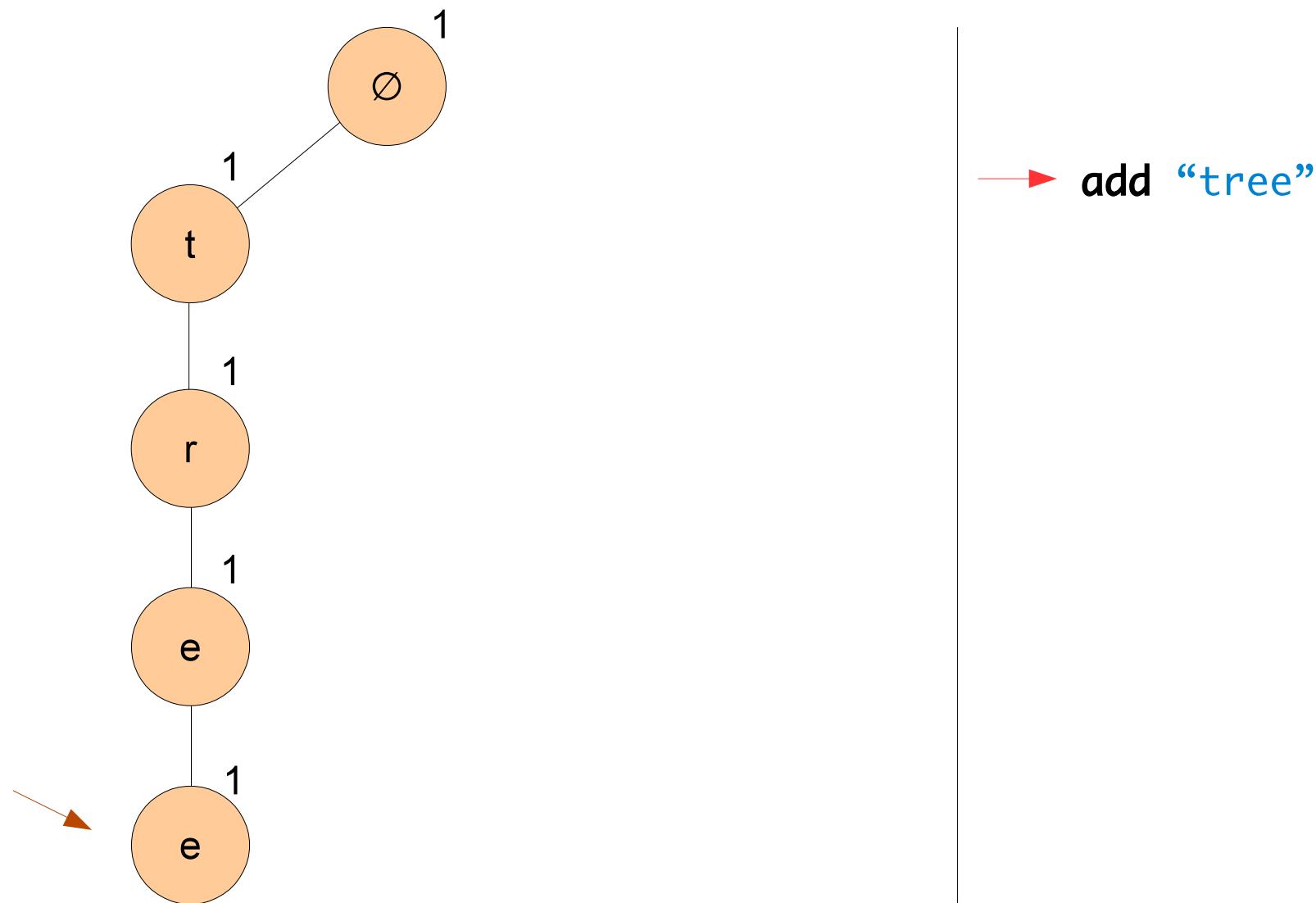
Tries: Παράδειγμα Επέκτασης (2)

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

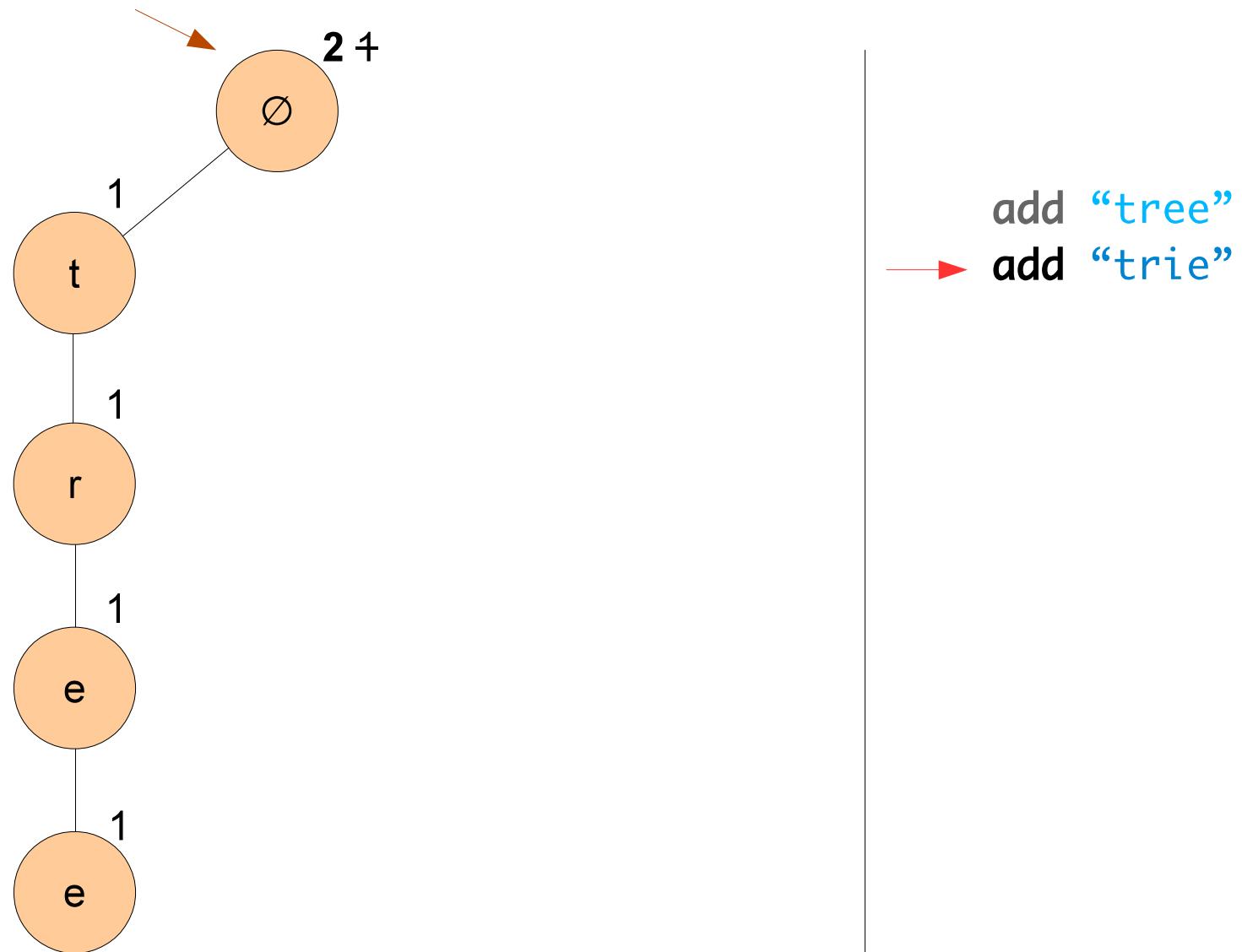
Υλοποίηση:

- 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.

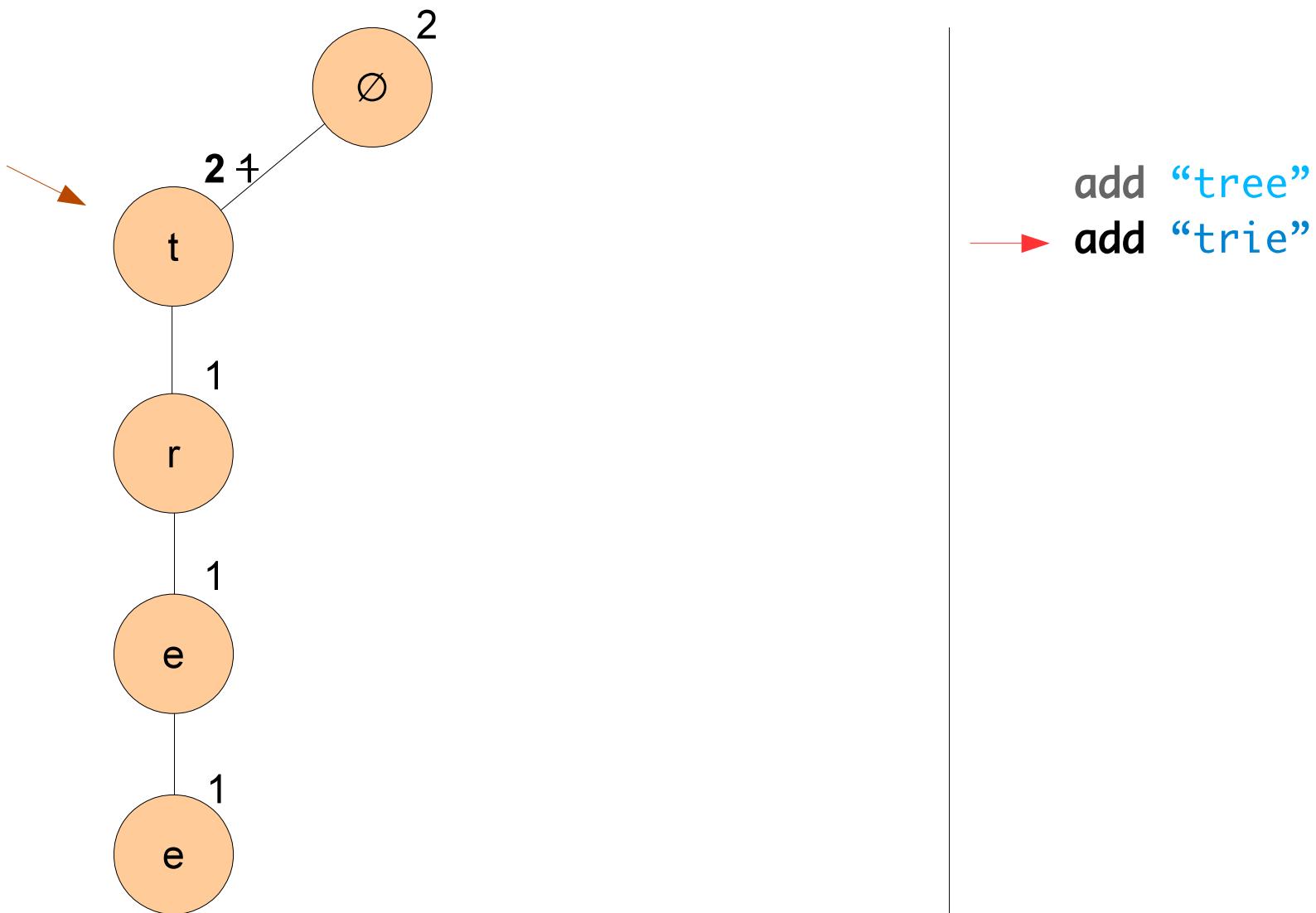
Tries: Παράδειγμα κατασκευής



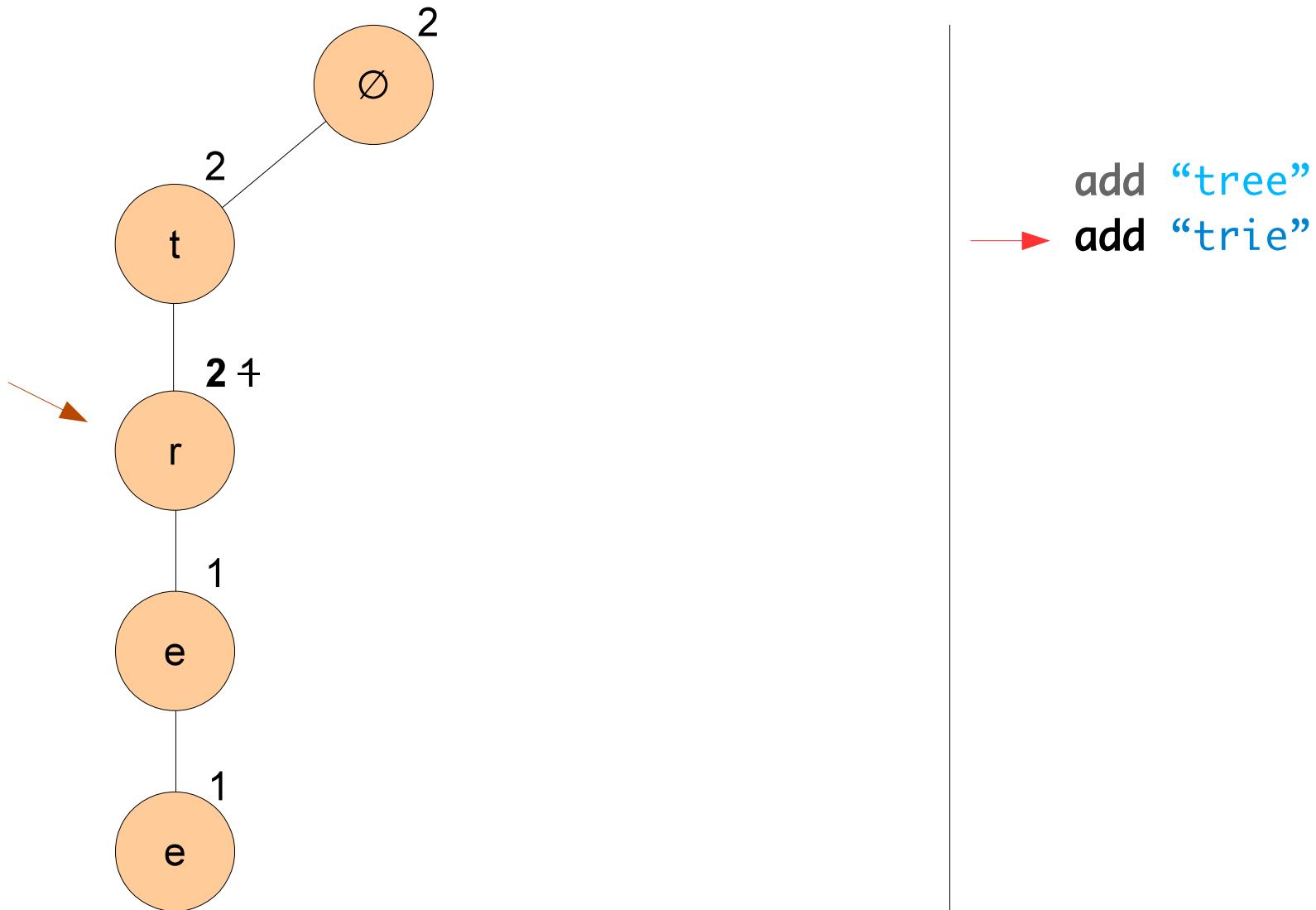
Tries: Παράδειγμα κατασκευής



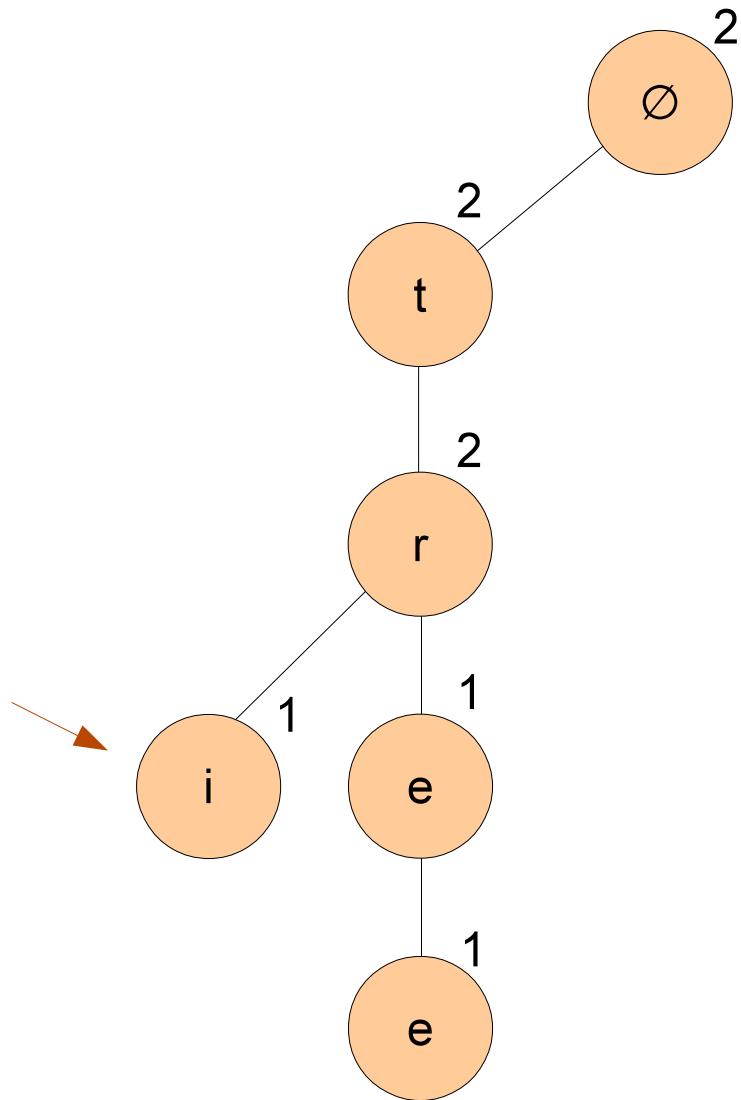
Tries: Παράδειγμα κατασκευής



Tries: Παράδειγμα κατασκευής

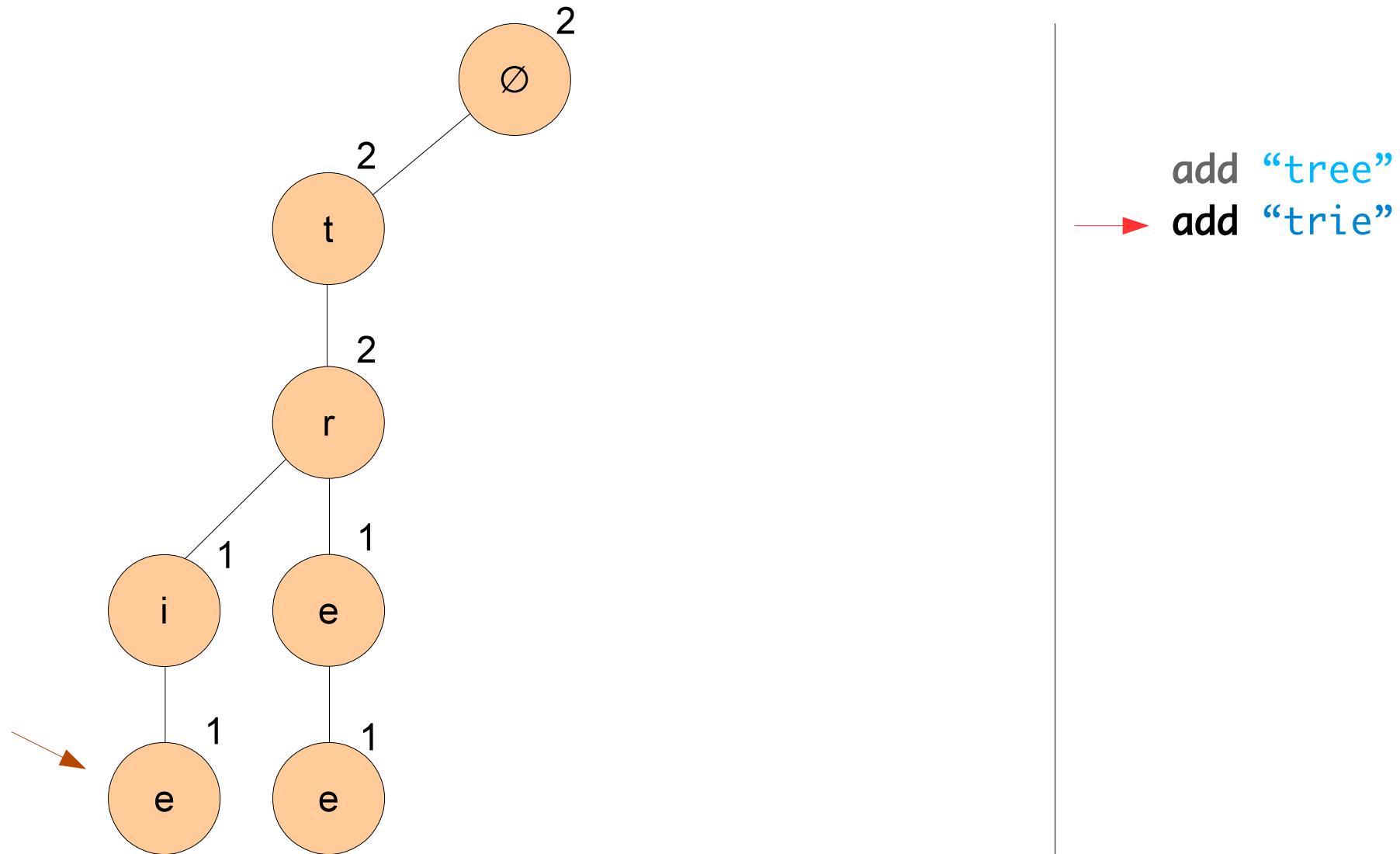


Tries: Παράδειγμα κατασκευής

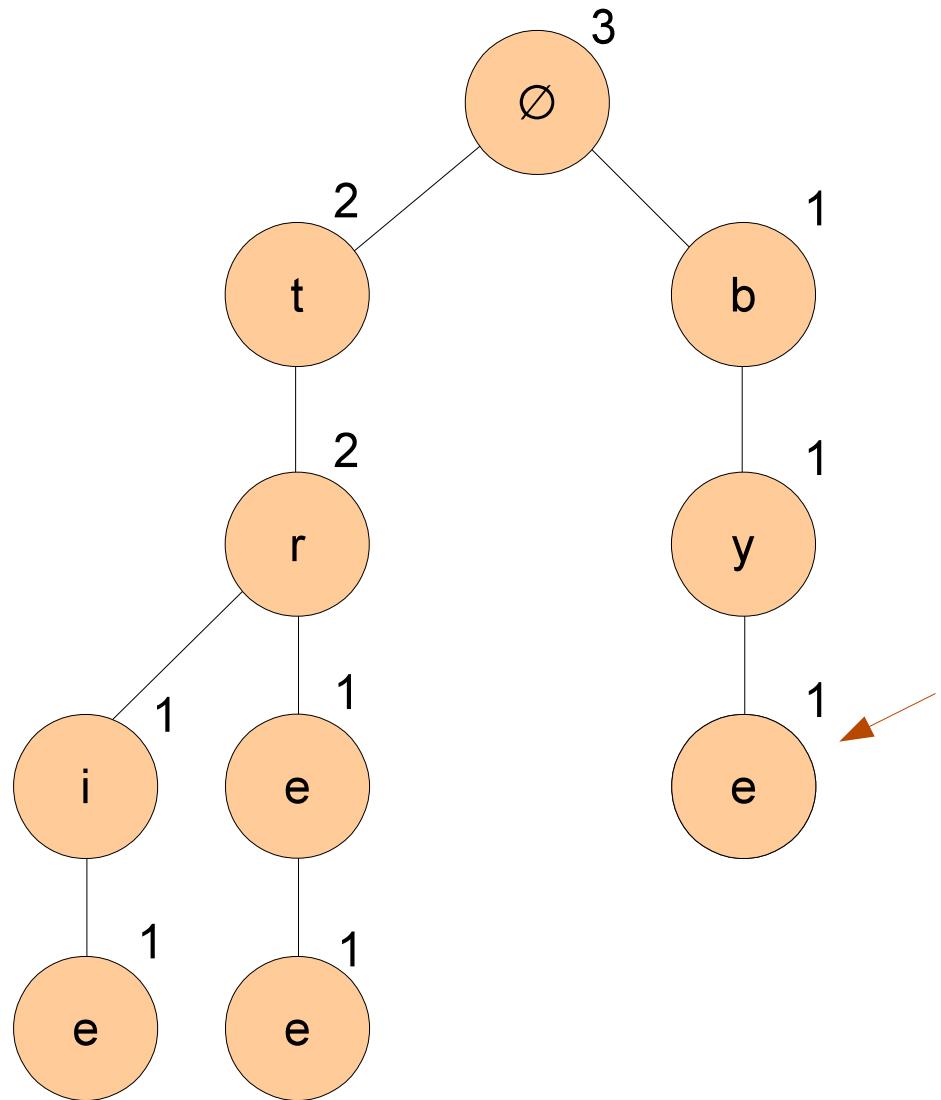


add “tree”
→ add “trie”

Tries: Παράδειγμα κατασκευής

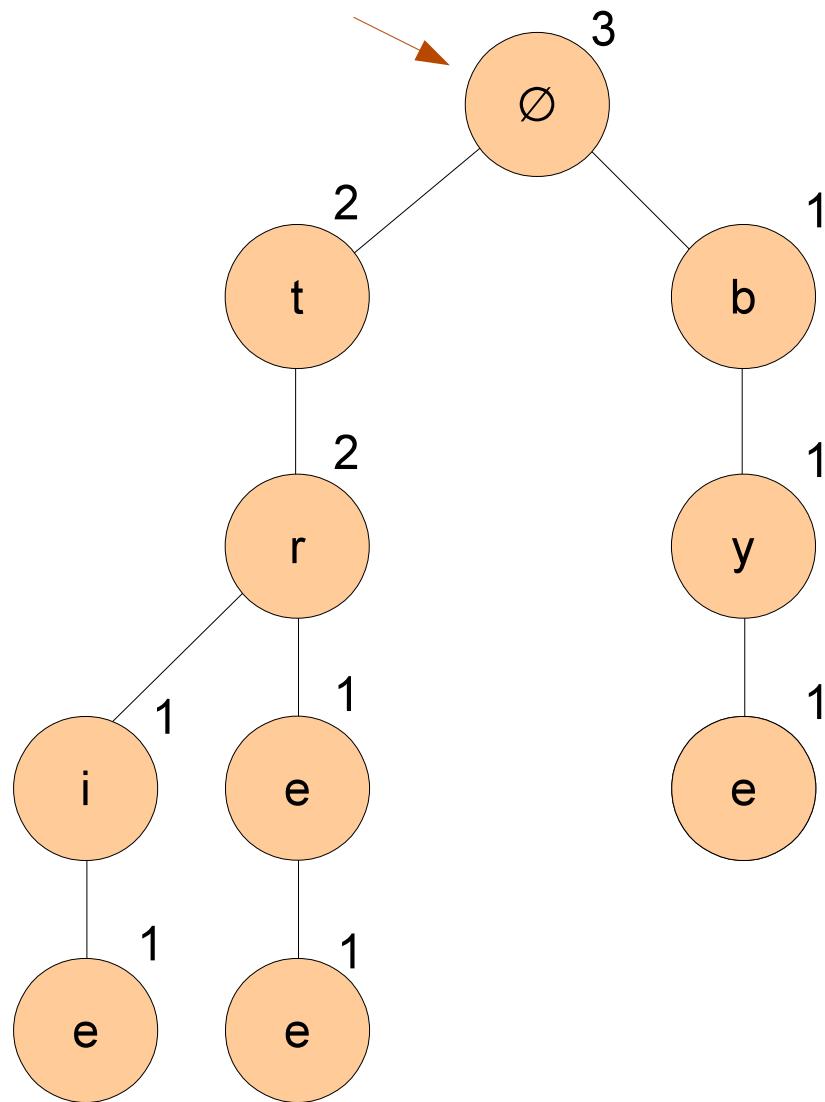


Tries: Παράδειγμα κατασκευής



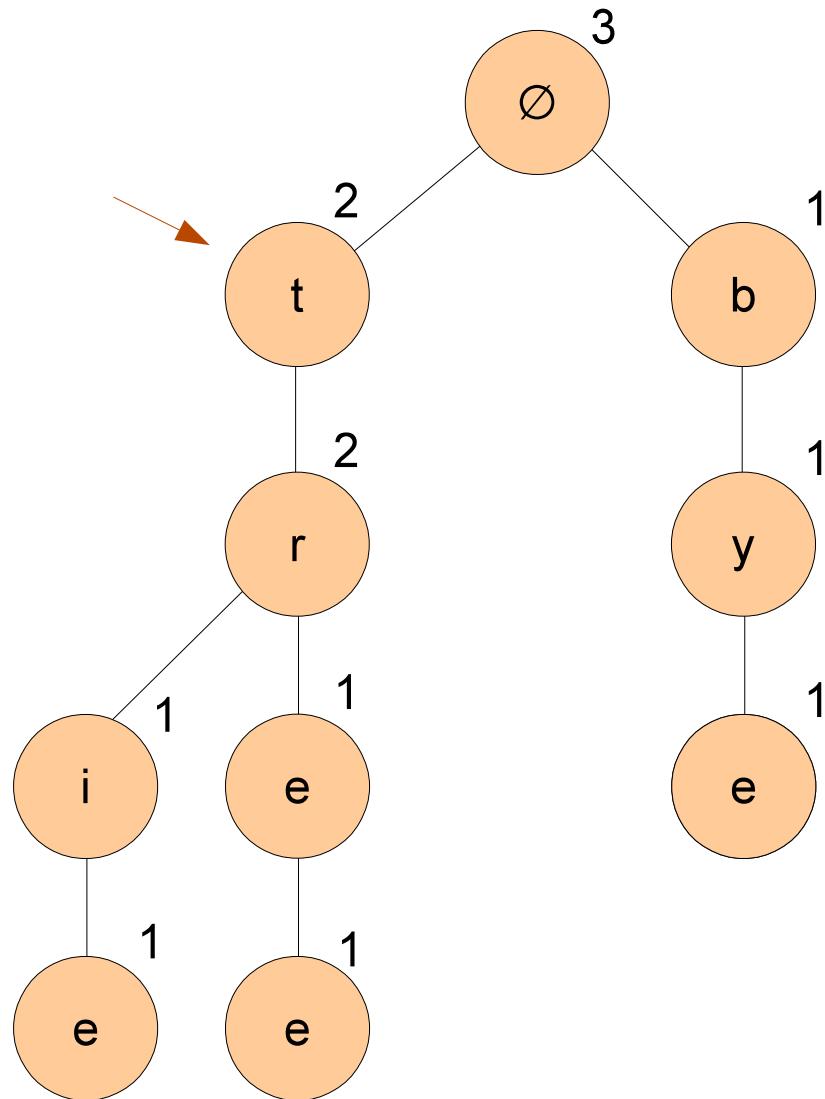
add “tree”
add “trie”
→ add “bye”

Tries: Παράδειγμα κατασκευής



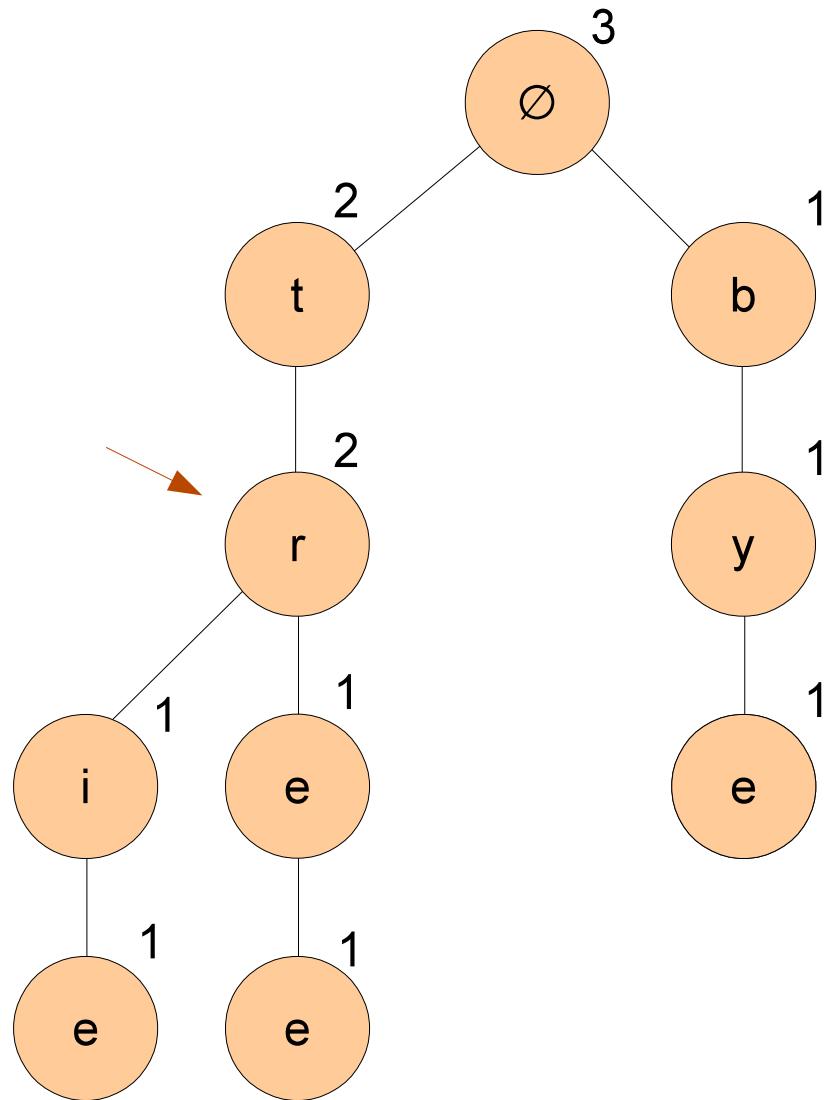
add “tree”
add “trie”
add “bye”
→ pcount “tr”

Tries: Παράδειγμα κατασκευής



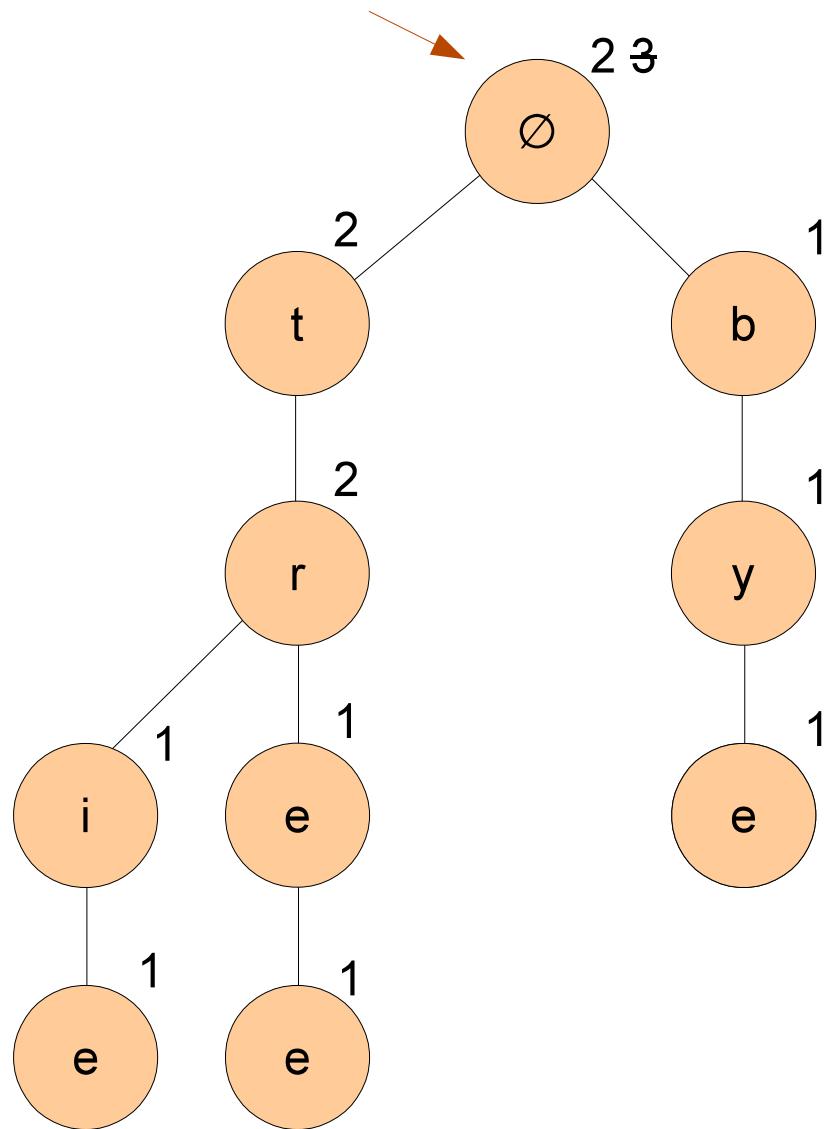
add “tree”
add “trie”
add “bye”
→ pcount “tr”

Tries: Παράδειγμα κατασκευής



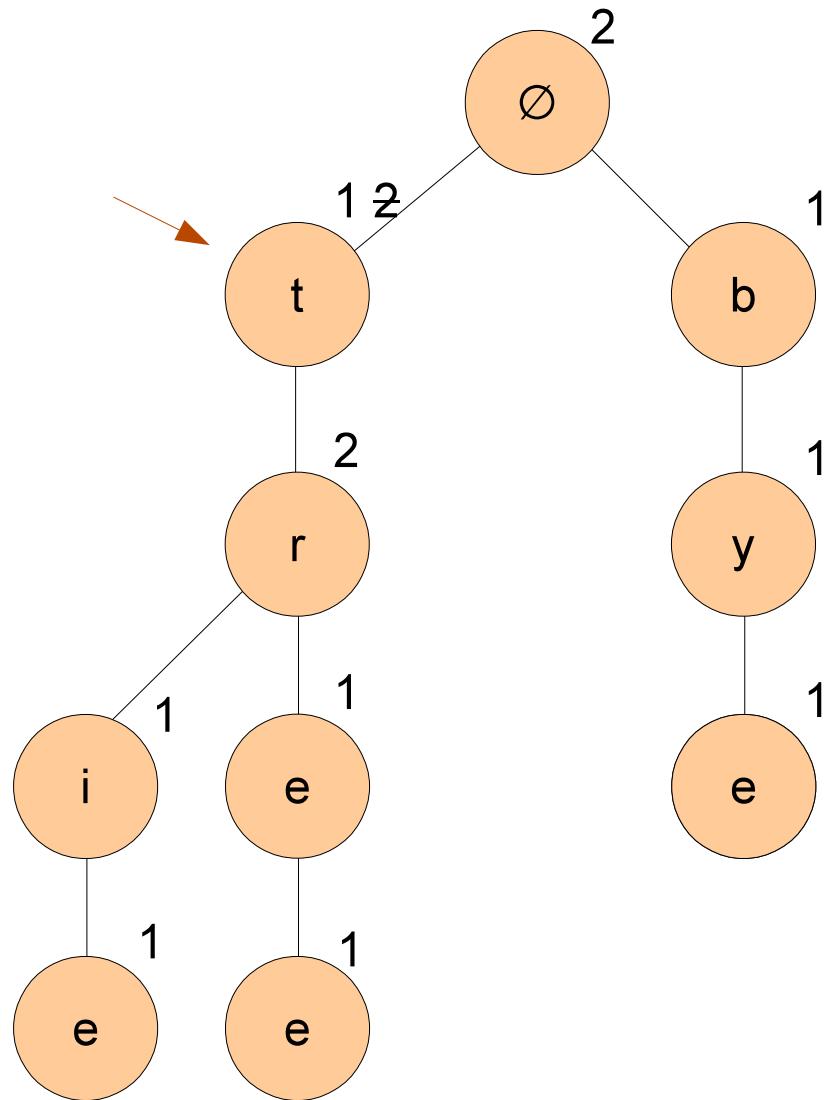
```
add "tree"
add "trie"
add "bye"
→ pcount "tr" 2
```

Tries: Παράδειγμα κατασκευής



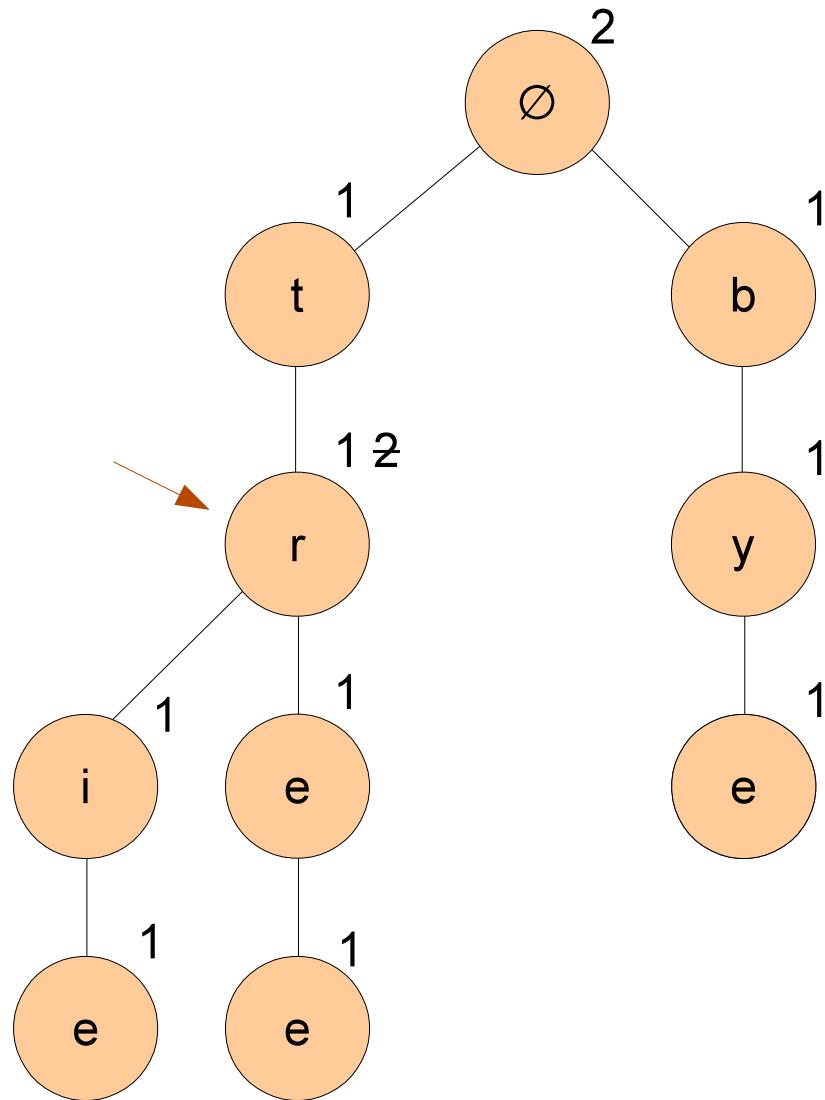
add "tree"
add "trie"
add "bye"
pcount "tr" 2
remove "trie"

Tries: Παράδειγμα κατασκευής



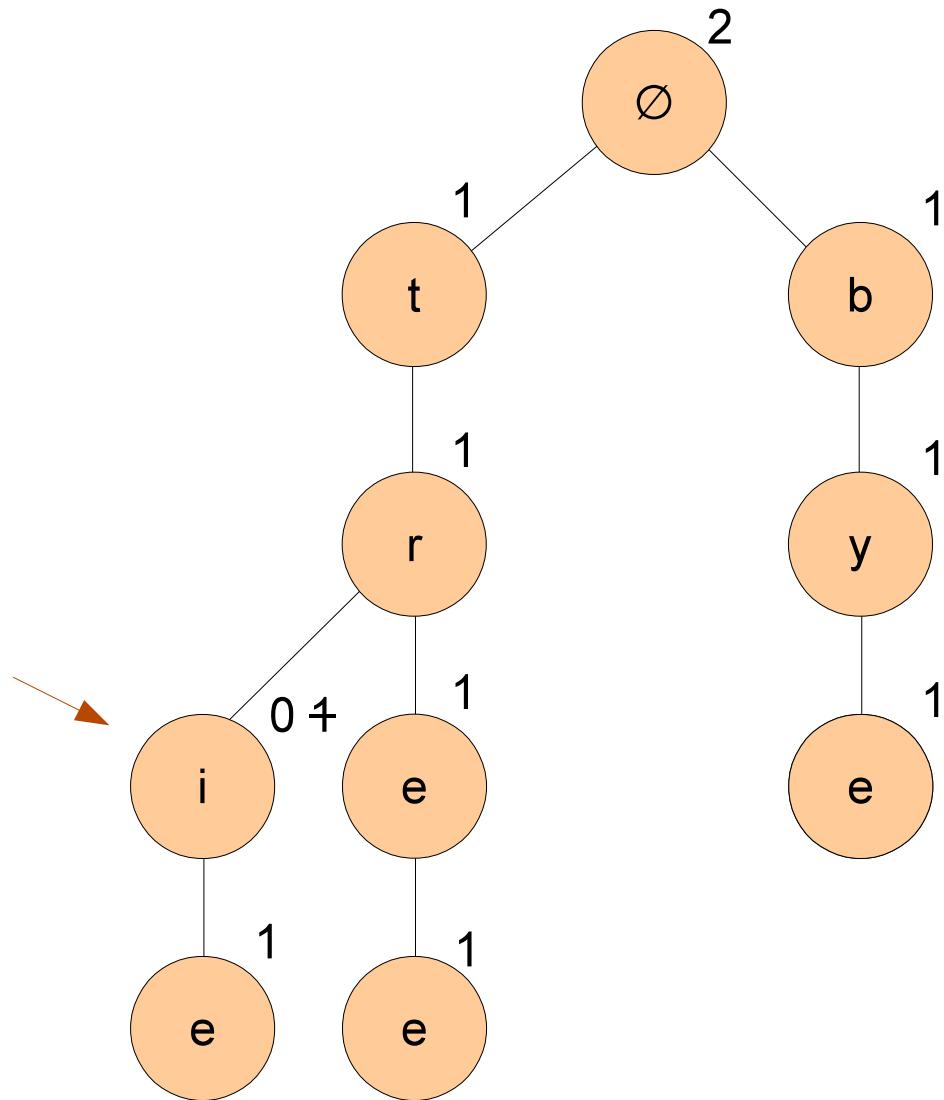
add "tree"
add "trie"
add "bye"
pcount "tr" 2
remove "trie"

Tries: Παράδειγμα κατασκευής



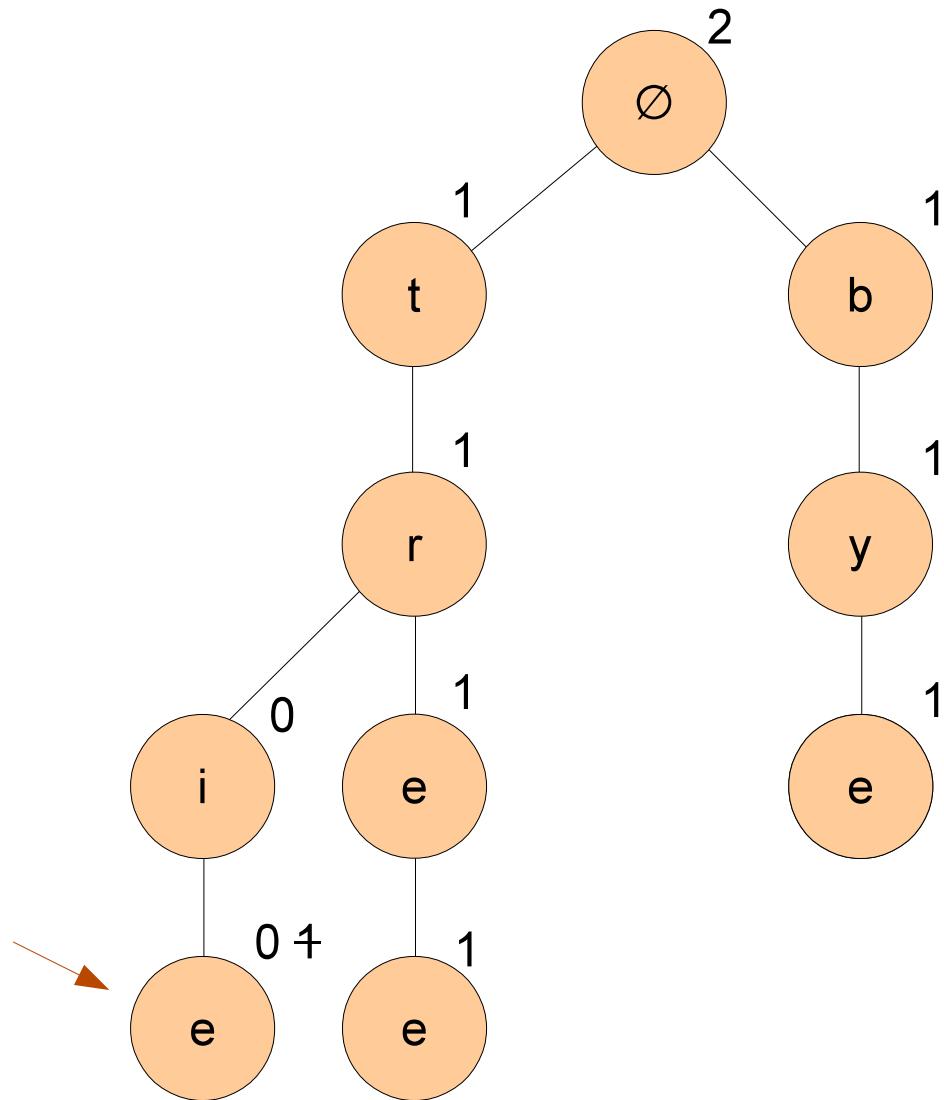
add "tree"
add "trie"
add "bye"
pcount "tr" 2
remove "trie"

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).