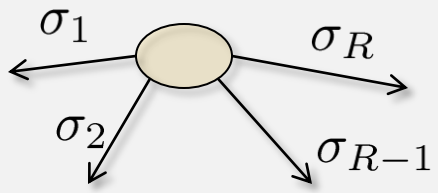
Ψηφιακά Δένδρα

Μελετάμε την περίπτωση όπου αποθηκεύουμε ένα (δυναμικό) σύνολο στοιχειών S τα οποία είναι ακολουθίες l συμβόλων από ένα πεπερασμένο αλφάβητο $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_R\}$

Ένα στοιχείο $x \in S$ γράφεται ως $x = x_1 x_2 \ldots x_l$, όπου κάθε $x_i \in \Sigma$.

Μπορούμε να χρησιμοποιήσουμε την παραπάνω αναπαράσταση για να αποθηκεύσουμε τα στοιχεία σε ένα R-αδικό δένδρο :



Ένας εσωτερικός κόμβος έχει μία εξερχόμενη ακμή για κάθε σύμβολο $\sigma_i \in \Sigma$

Tries

Ένα trie (από re**trie**val) είναι ένα ψηφιακό δένδρο όπου :

- Τα στοιχεία αποθηκεύονται (ολόκληρα ή μέρος τους) στα φύλλα του δένδρου.
- Οι ακμές του μονοπατιού από τη ρίζα προς το φύλλο που αποθηκεύει ένα στοιχείο $x = x_1 x_2 \dots x_l$ σχηματίζουν ένα πρόθεμα $x_1 x_2 \dots x_k,\ k < l$.

Θα εστιάσουμε πρώτα στην περίπτωση $\Sigma = \{0, 1\}\ (R = 2)$.

Αναδρομικός ορισμός

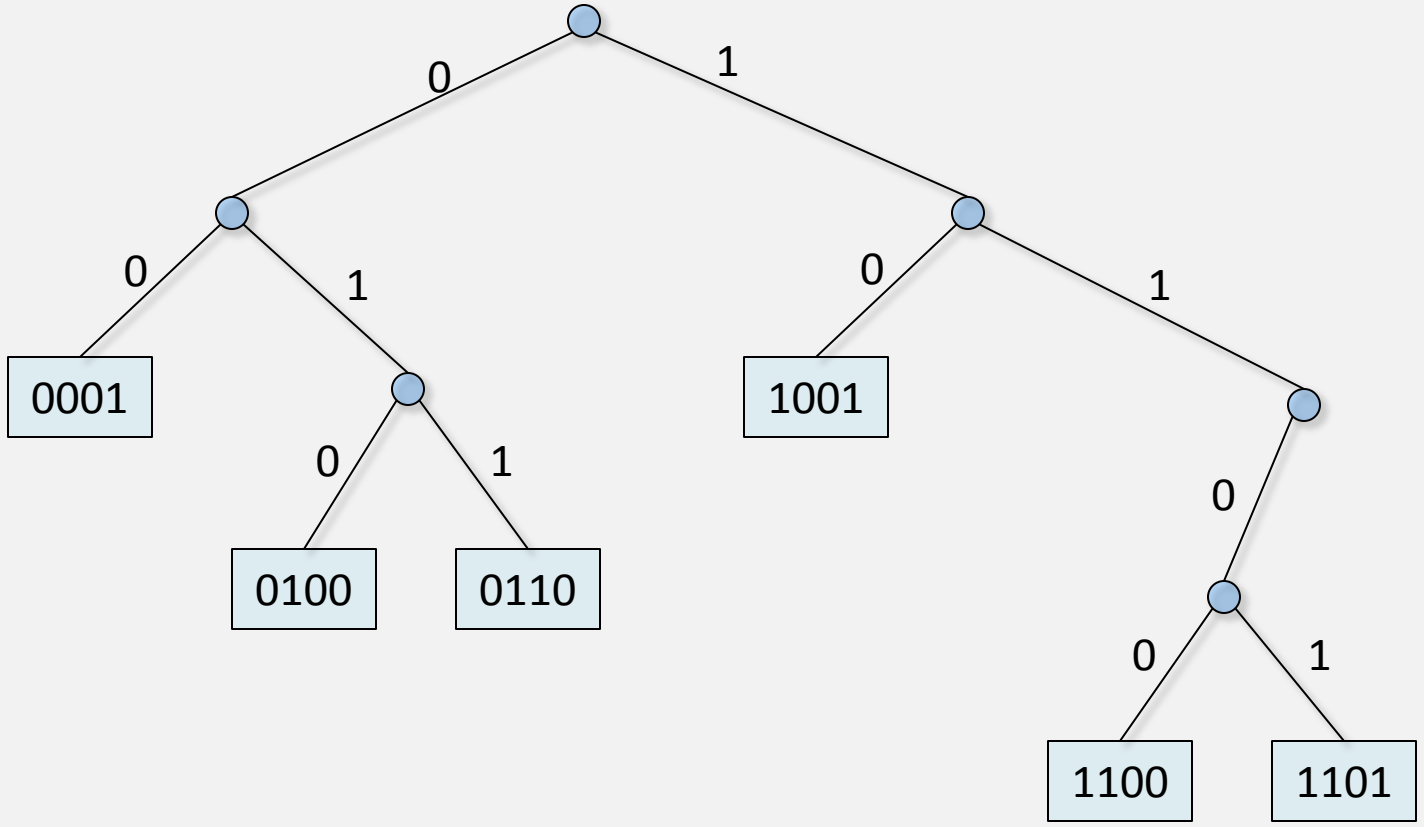
Ένα trie με n στοιχεία είναι ένα δυαδικό δένδρο το οποίο αποθηκεύει στοιχεία στα φύλλα του και ορίζεται αναδρομικά ως εξής :

- $n = 0$: το trie είναι ένας κενός κόμβος.
- $n = 1$: το trie είναι ένα φύλλο που περιέχει το μοναδικό στοιχείο.
- $n > 1$: το trie αποτελείται από ένα εσωτερικό κόμβο ρίζα του οποίου ο αριστερός σύνδεσμος αναφέρεται σε trie που περιέχει τα στοιχεία με το πρώτο ψηφίο 0 και ο δεξιός σύνδεσμος αναφέρεται σε trie που περιέχει τα στοιχεία με το πρώτο ψηφίο 1 .

Το πρώτο ψηφίο αφαιρείται για την κατασκευή των υποδένδρων.

Tries

Αναζήτηση στοιχείου x: Ξεκινάμε από τη ρίζα και ακολουθούμε τις ακμές που αντιστοιχούν στα ψηφία του x μέχρι να καταλήξουμε σε φύλλο ή κενό κόμβο.



Tries

Εισαγωγή στοιχείου $x = x_1 x_2 \ldots x_l$

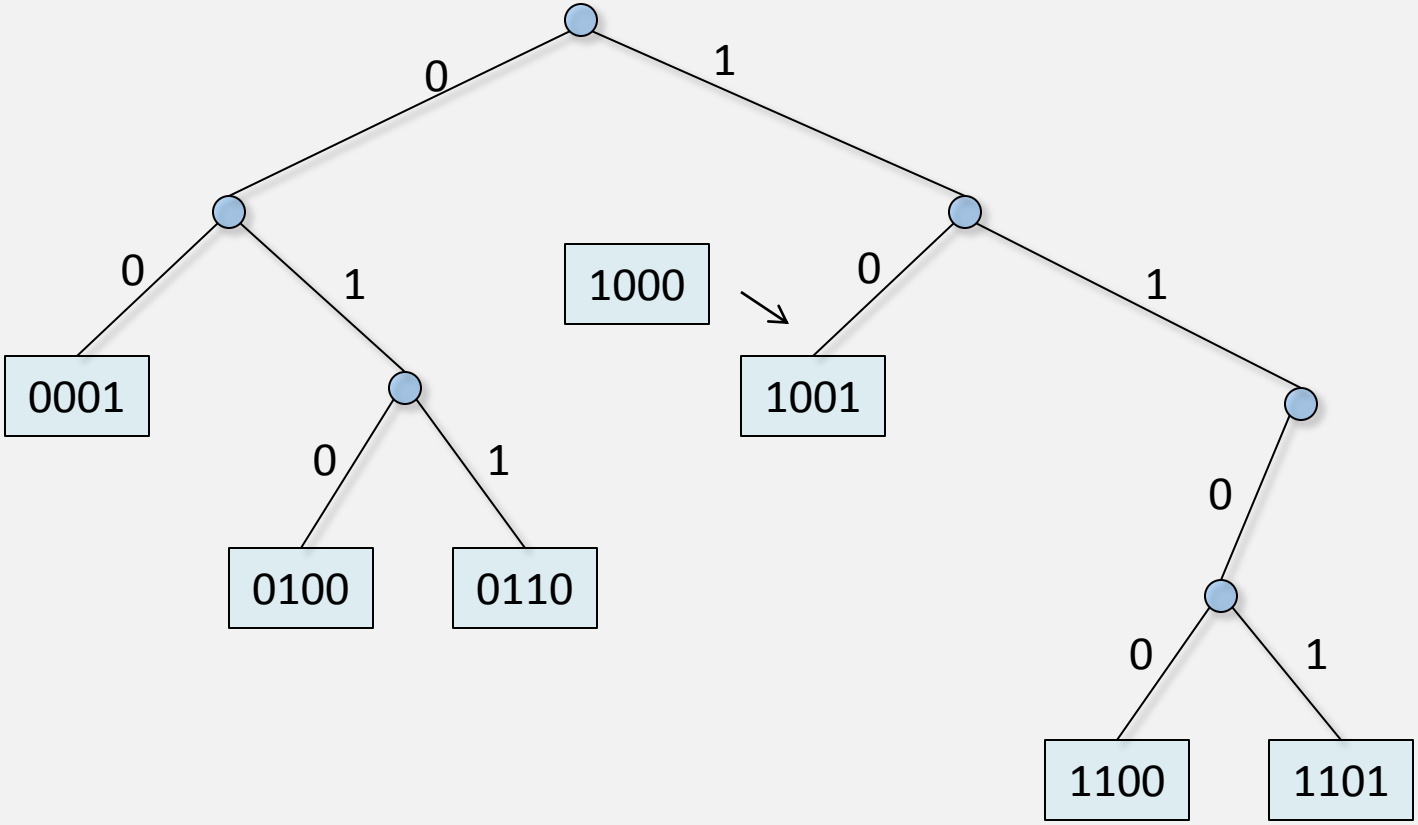
Εκτελούμε τη διαδικασία αναζήτησης του x. Έστω v ο κόμβος στον οποίο καταλήγει η αναζήτηση. Έστω k αριθμός των ακμών που ακολούθησε η αναζήτηση και έστω y το στοιχείο που περιέχει ο v.

- Αν ο v είναι ο κενός κόμβος τότε απλά εισάγουμε στη θέση του νέο κόμβο που αποθηκεύει το στοιχείο x.
- Διαφορετικά έστω ότι $y_i = x_i$ για $i = 1, 2, \ldots, j$, όπου $j \geq k$, και $y_{j+1} \neq x_{j+1}$. Εισάγουμε στη θέση του v ένα μονοπάτι που αντιστοιχεί στα ψηφία των θέσεων από k έως j. Τοποθετούμε τους κόμβους που περιέχουν τα στοιχεία x και y ως παιδιά του τελευταίου κόμβου του μονοπατιού.

Tries

Παράδειγμα : Εισαγωγή 1000

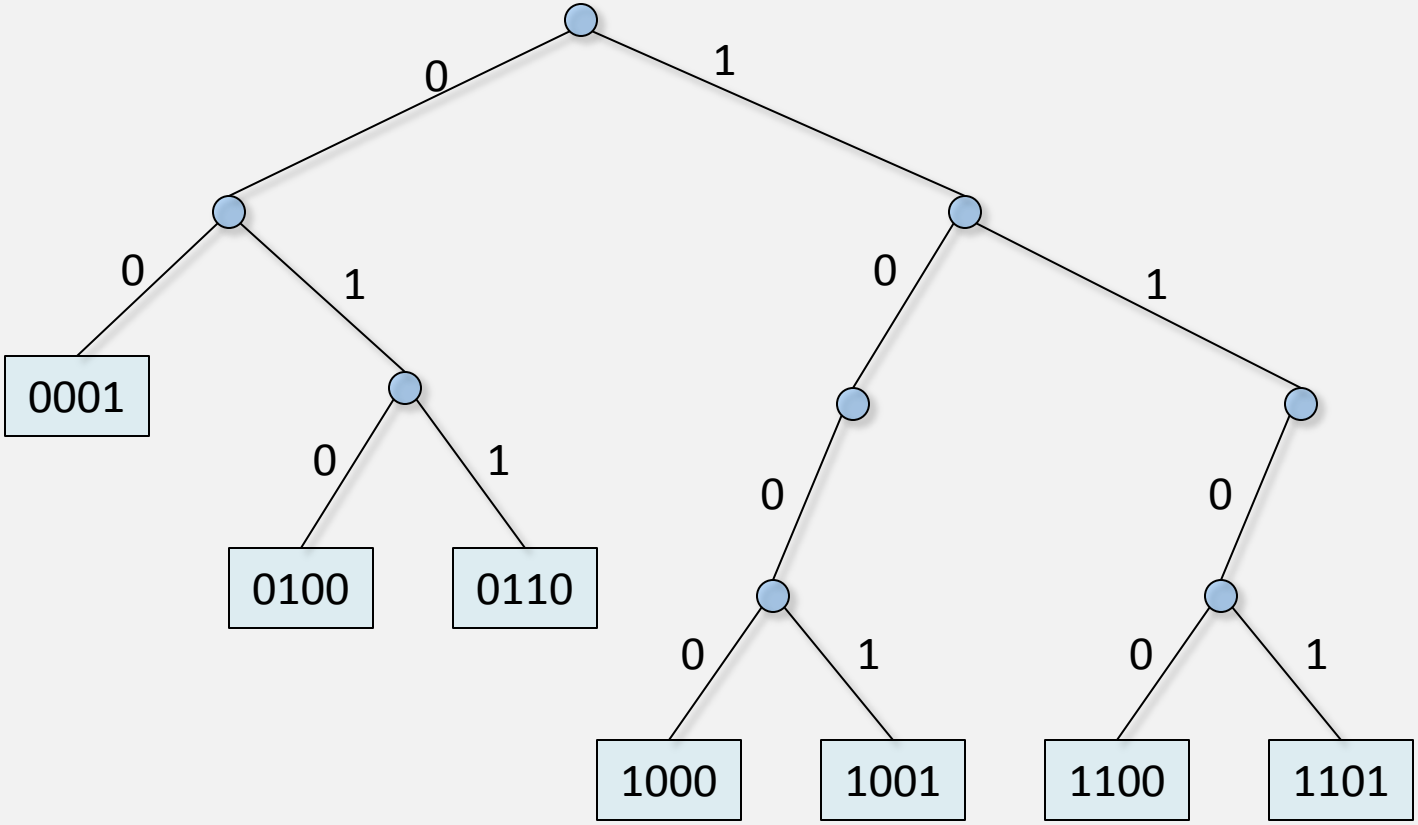
$l = 4$



Tries

Παράδειγμα : Εισαγωγή 1000

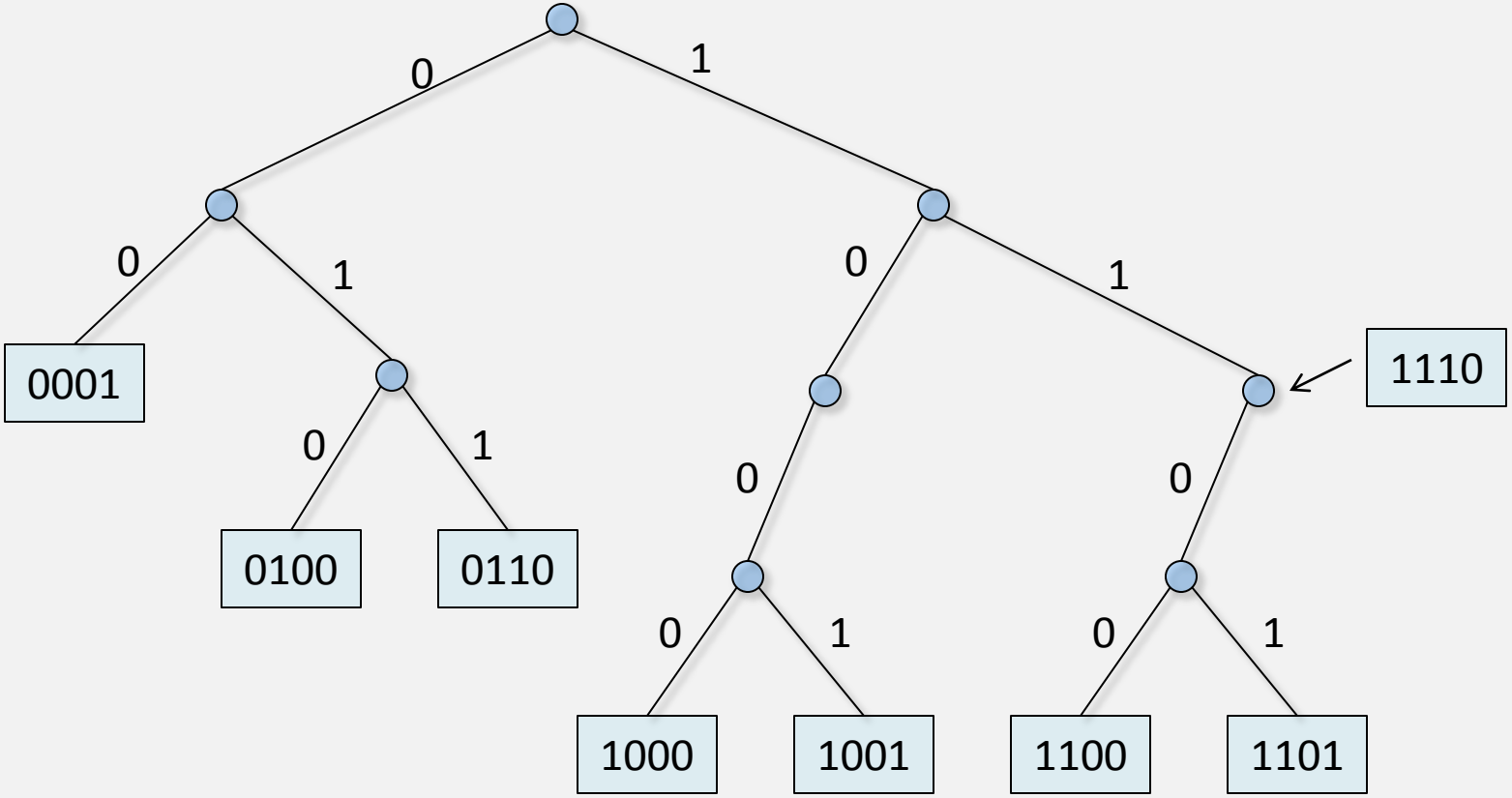
$l = 4$



Tries

Παράδειγμα : Εισαγωγή 1101

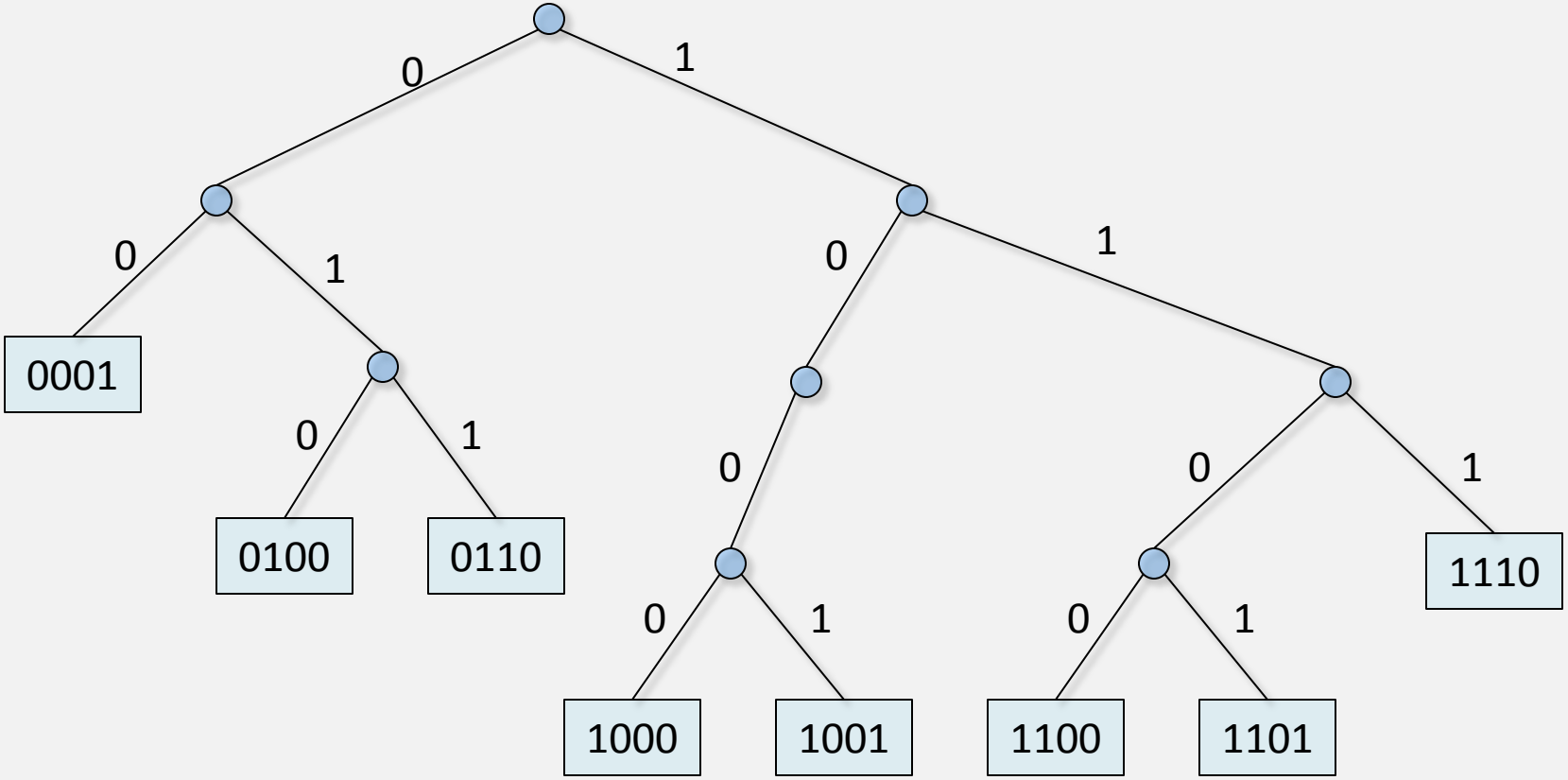
$l = 4$



Tries

Παράδειγμα : Εισαγωγή 1101

$l = 4$



Tries

Ιδιότητες

- Η μορφή του trie είναι ανεξάρτητη από τη σειρά εισαγωγής των στοιχείων. Κάθε δεδομένο σύνολο διακριτών στοιχείων δημιουργεί ένα μοναδικό trie.

- Η αναζήτηση ή εισαγωγή απαιτεί χρόνο $O(l)$ στη χειρότερη περίπτωση.

- Η αναζήτηση ή εισαγωγή ενός τυχαίου στοιχείου σε trie κατασκευασμένο από n τυχαίες (διακεκριμένες) ακολουθίες ψηφίων, απαιτεί κατά μέσο όρο χρόνο $O(\log n)$.

- Ένα trie κατασκευασμένο από n τυχαίες (διακεκριμένες) ακολουθίες ψηφίων περιέχει κατά μέσο όρο $n/\ln 2 \approx 1.44n$ εσωτερικούς κόμβους.

Tries

- Η αναζήτηση ή εισαγωγή ενός τυχαίου στοιχείου σε trie κατασκευασμένο από n τυχαίες (διακεκριμένες) ακολουθίες ψηφίων, απαιτεί κατά μέσο όρο χρόνο $O(\log n)$.

Απόδειξη

Έστω x ένα τυχαίο στοιχείο. Η πιθανότητα που έχει καθένα από τα n στοιχεία του τυχαίου trie να διαφέρει με το x σε τουλάχιστον ένα από τα πρώτα t ψηφία είναι $(1-1/2^t)^n$. Άρα η πιθανότητα το x να ταιριάζει σε όλα τα t πρώτα ψηφία με κάποιο στοιχείο του trie είναι

$$1-\left(1-\frac{1}{2^t}\right)^n \sim 1-e^{-n/2^t}$$

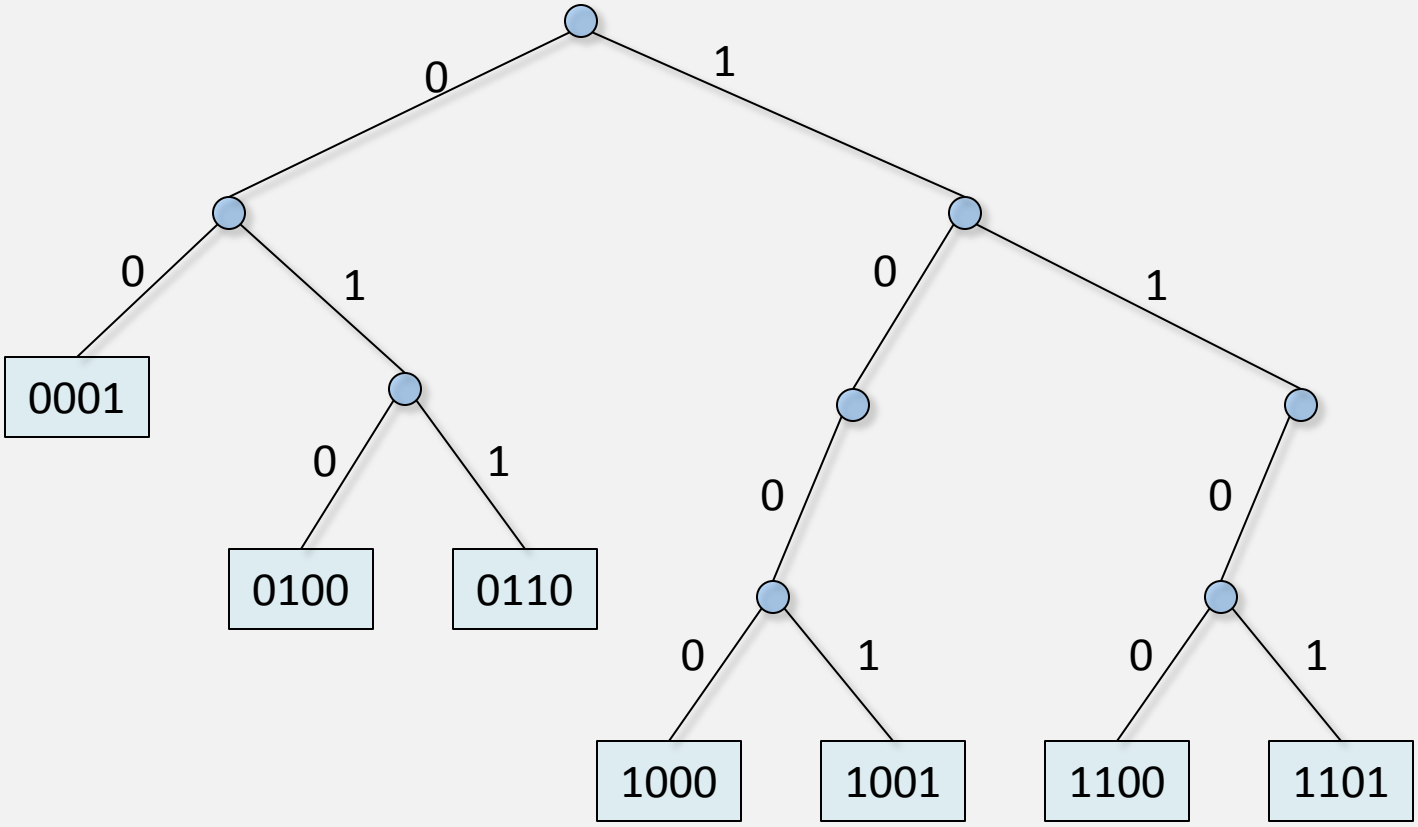
όπου χρησιμοποιήσαμε την προσέγγιση $(1-1/x)^x \sim e^{-1}$.

Άρα ο μέσος χρόνος αναζήτησης του x είναι

$$\sum_{t=0}^{n}\left(1-e^{-n/2^t}\right)=O(\log n)$$

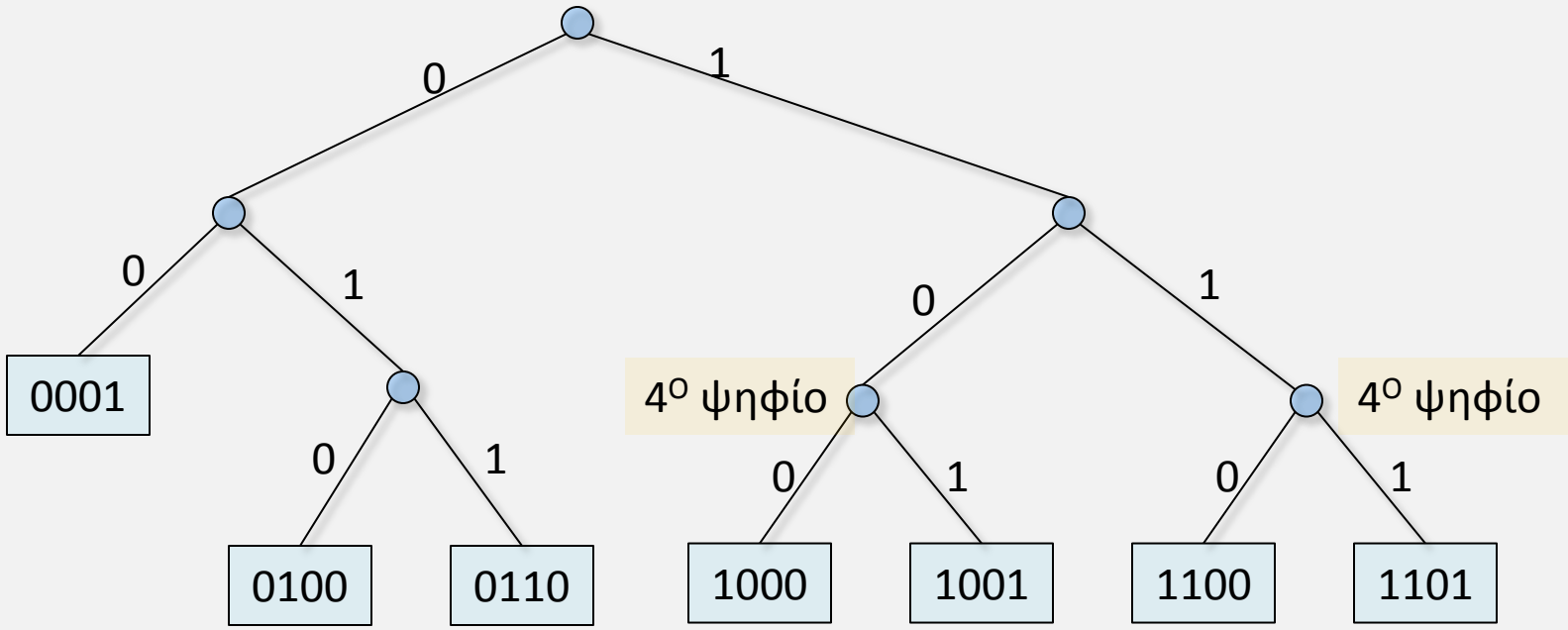
Συμπιεσμένα Tries

Για να αποφύγουμε τη σπατάλη χώρου σε ένα trie μπορούμε να συμπιέσουμε τους κόμβους που έχουν μονόδρομη διακλάδωση:



Συμπιεσμένα Tries

Για να αποφύγουμε τη σπατάλη χώρου σε ένα trie μπορούμε να συμπιέσουμε τους κόμβους που έχουν μονόδρομη διακλάδωση:



Σε κάθε κόμβο τοποθετούμε τον αριθμοδείκτη του ψηφίου που πρόκειται να ελεγχθεί.

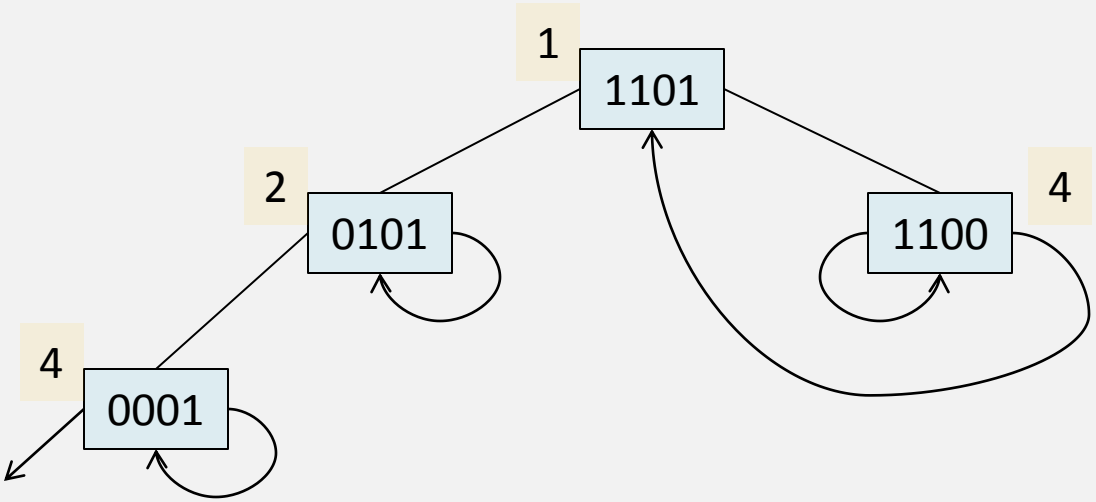
Επιτυγχάνει $O(n)$ χώρο αλλά κάνει δυσκολότερη την εισαγωγή.

Patricia Tries

"**P**ractical **a**lgorithm **t**o **r**etrieve **i**nformation **c**oded **i**n **a**lphanumeric"

Συμπιεσμένα tries που επιτυγχάνουν γρήγορη εισαγωγή.

- Τα στοιχεία αποθηκεύονται στους εσωτερικούς κόμβους αλλά δε χρησιμοποιούνται κατά την αναζήτηση.
- Αντικαθιστούμε τους συνδέσμους προς τα φύλλα με συνδέσμους οι οποίοι δείχνουν προς τα επάνω, στο σωστό εσωτερικό κόμβο του trie.
- Η αναζήτηση γίνεται όπως στο συμπιεσμένο trie, μόνο που ολοκληρώνεται στον πρώτο κόμβο που συναντάμε μέσω συνδέσμου προς τα πάνω.



Patricia Tries

Εισαγωγή στοιχείου $x = x_1 x_2 \ldots x_l$

Εκτελούμε τη διαδικασία αναζήτησης του x . Έστω v ο κόμβος στον οποίο καταλήγει η αναζήτηση. Έστω y το στοιχείο που αποθηκεύει ο v, και έστω $j+1$ η πρώτη θέση στην οποία διαφωνούν τα ψηφία των x και y .
($y_i = x_i$ για $i = 1, 2, \ldots, j$.)

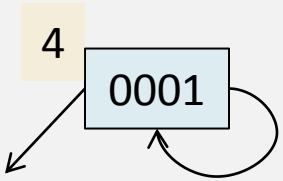
Ξεκινάμε ξανά από τη ρίζα και ελέγχουμε τους αριθμοδείκτες των κόμβων στο μονοπάτι αναζήτησης :

- Αν δεν υπάρχει κόμβος με αριθμοδείκτη $> j+1$ τότε προσθέτουμε ένα νέο κόμβο που διακρίνει το στοιχείο x από το y .

- Διαφορετικά εισάγουμε ένα νέο κόμβο που ελέγχει το ψηφίο $j+1$.

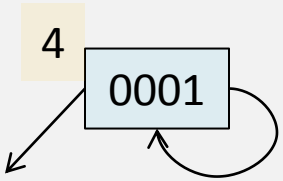
Patricia Tries

Παράδειγμα : Εισαγωγή 0001



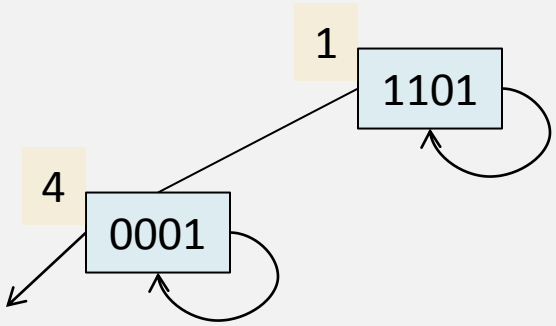
Patricia Tries

Παράδειγμα : Εισαγωγή 1101



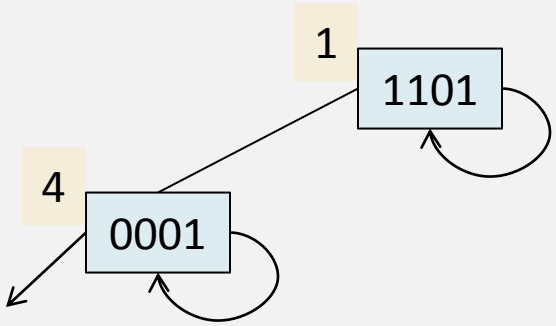
Patricia Tries

Παράδειγμα : Εισαγωγή 1101



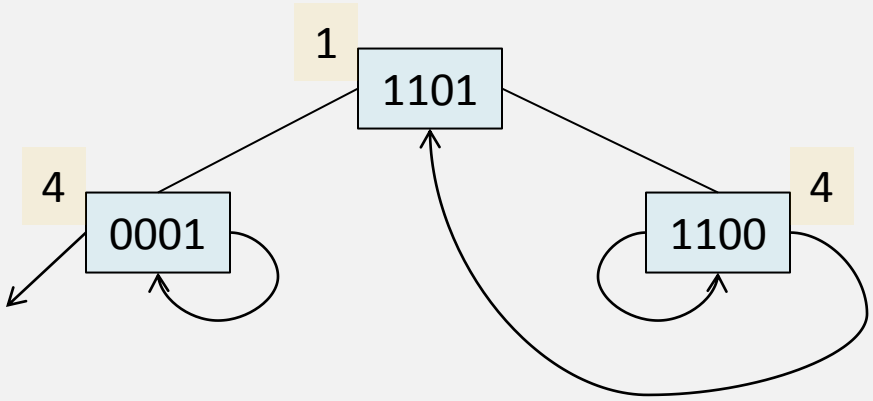
Patricia Tries

Παράδειγμα : Εισαγωγή 1100



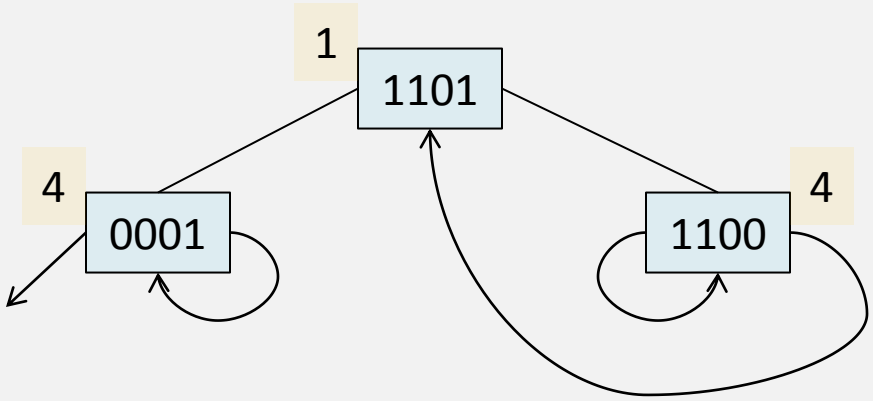
Patricia Tries

Παράδειγμα : Εισαγωγή 1100



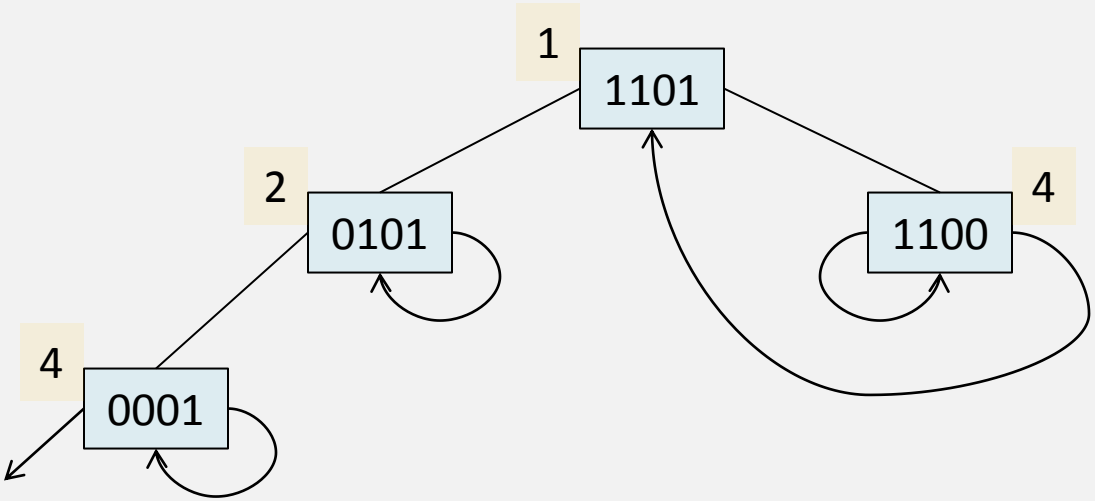
Patricia Tries

Παράδειγμα : Εισαγωγή 0101



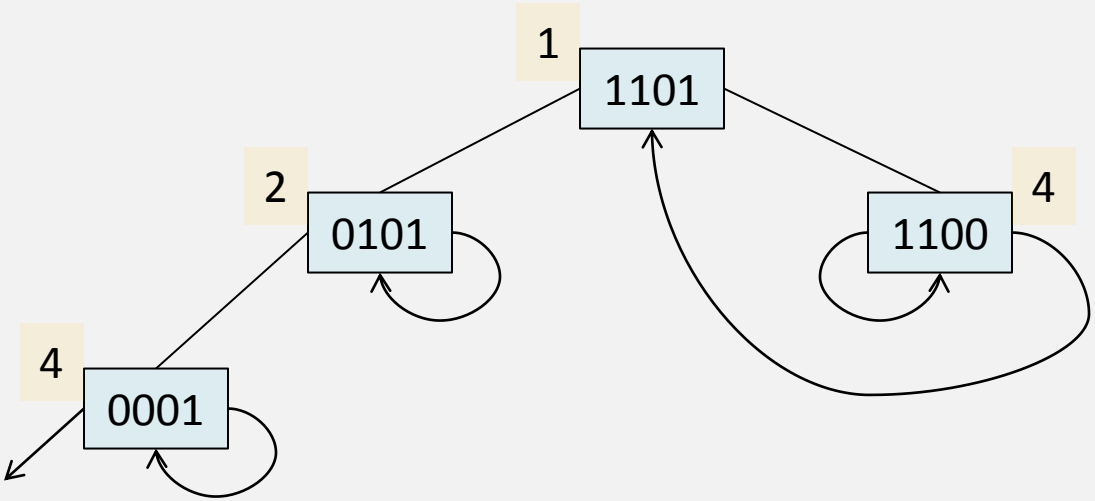
Patricia Tries

Παράδειγμα : Εισαγωγή 0101



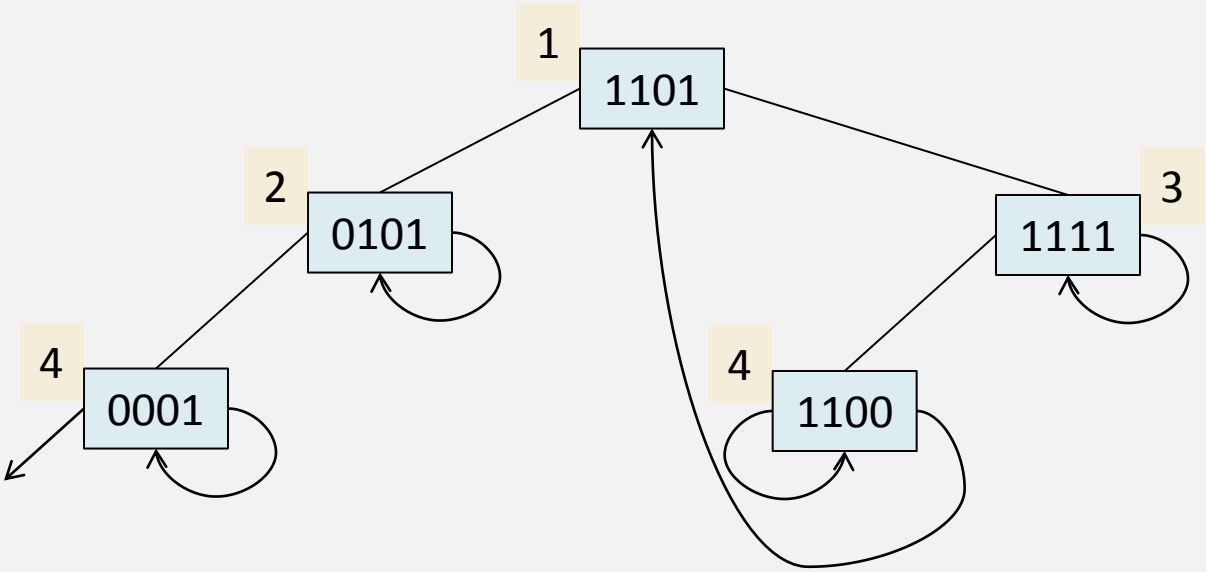
Patricia Tries

Παράδειγμα : Εισαγωγή 1111



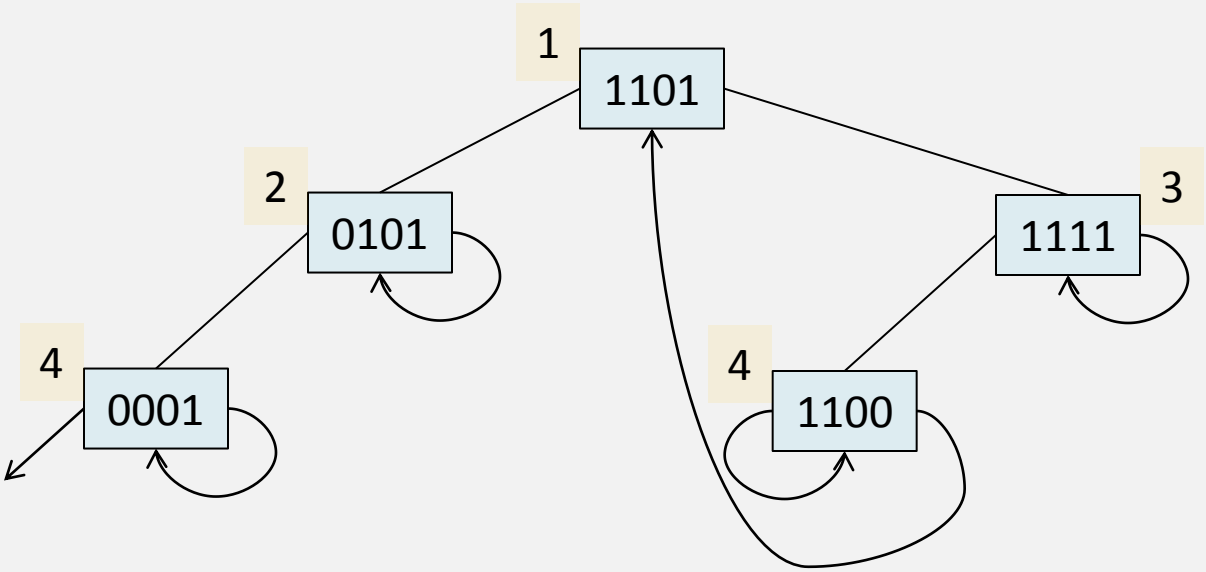
Patricia Tries

Παράδειγμα : Εισαγωγή 1111



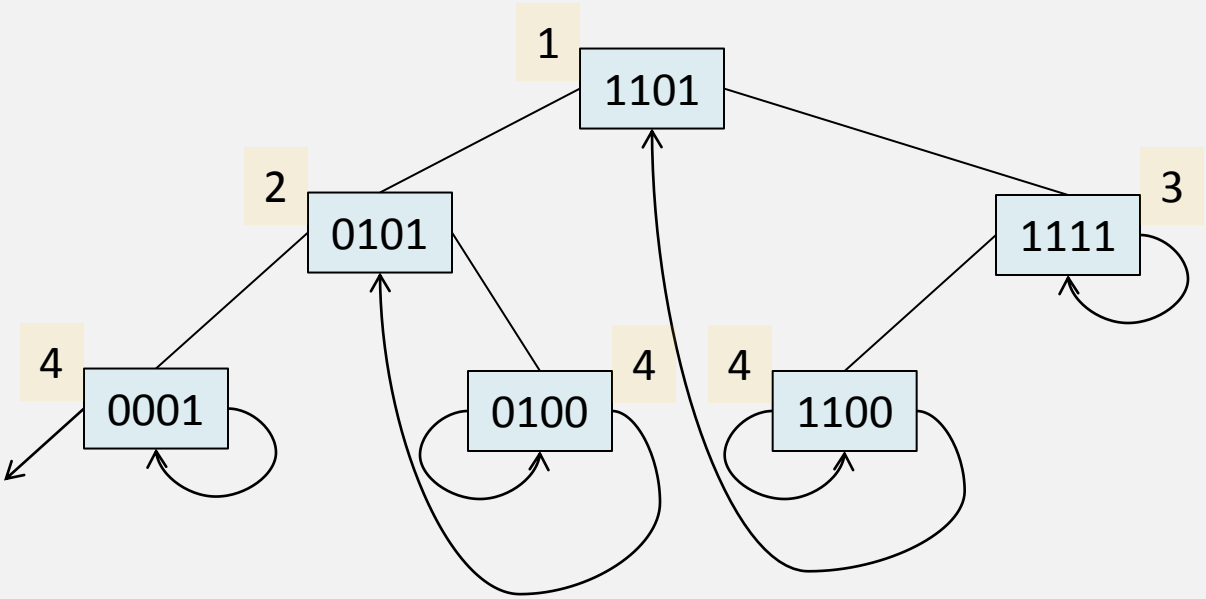
Patricia Tries

Παράδειγμα : Εισαγωγή 0100



Patricia Tries

Παράδειγμα : Εισαγωγή 0100



Patricia Tries

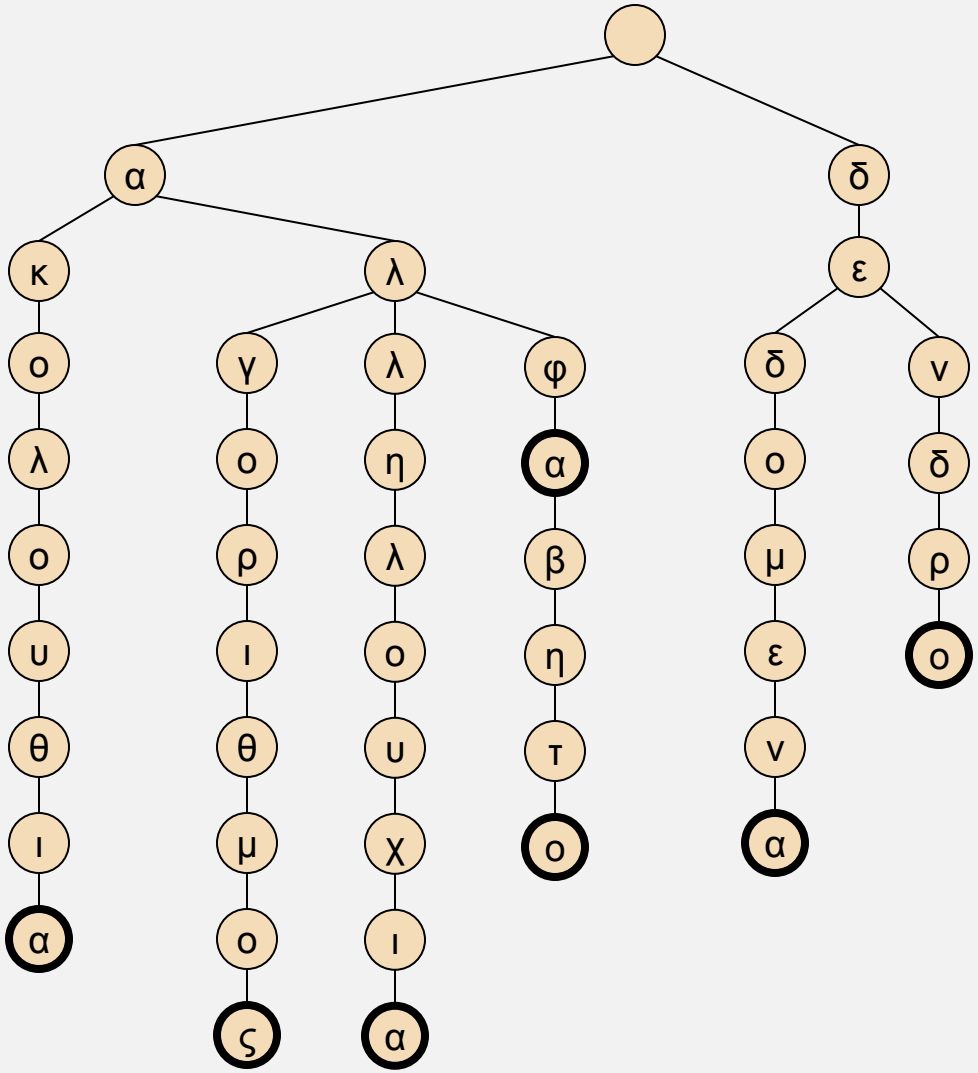
Ιδιότητα

Η αναζήτηση ή εισαγωγή ενός τυχαίου στοιχείου σε trie κατασκευασμένο από n τυχαίες (διακεκριμένες) ακολουθίες ψηφίων, απαιτεί περίπου $\lg n$ συγκρίσεις κατά μέσο όρο και περίπου $2 \lg n$ συγκρίσεις στη χειρότερη περίπτωση.

Tries για αποθήκευση αλφαριθμητικών

Αποθηκεύουμε τις λέξεις (αλφαριθμητικά) με έμμεσο τρόπο στους κόμβους του trie.

Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι

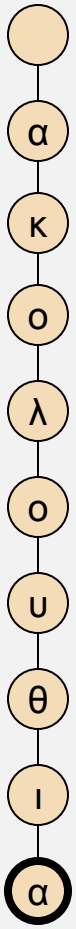


Tries για αποθήκευση αλφαριθμητικών

Παράδειγμα:

Trie για τις λέξεις

- ⇨ ακολουθία
- αλγόριθμος
- αλληλουχία
- αλφάβητο
- άλφα
- δεδομένα
- δένδρο



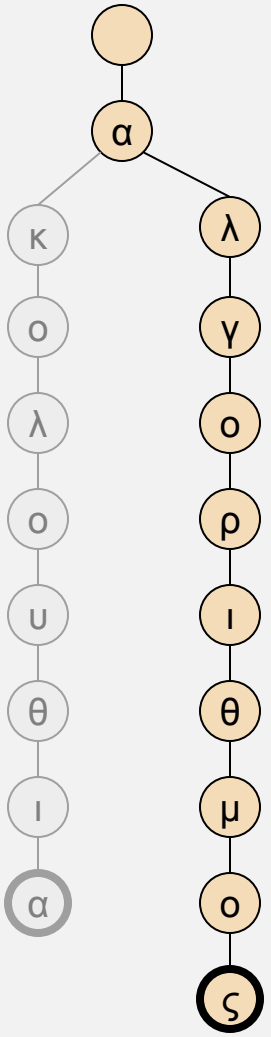
Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι 

Tries για αποθήκευση αλφαριθμητικών

Παράδειγμα:

Trie για τις λέξεις

- ακολουθία
- ⇨ αλγόριθμος
- αλληλουχία
- αλφάβητο
- άλφα
- δεδομένα
- δένδρο



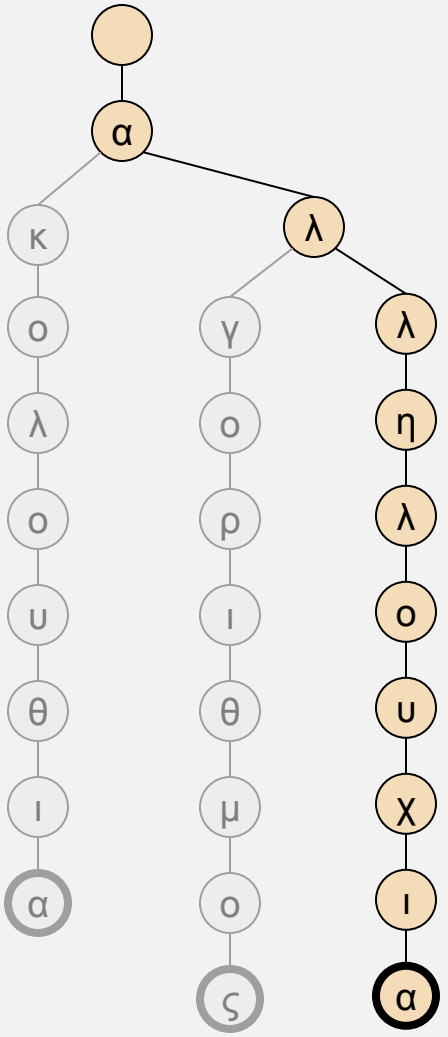
Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι

Tries για αποθήκευση αλφαριθμητικών

Παράδειγμα:

Trie για τις λέξεις

- ακολουθία
- αλγόριθμος
- ⇨ αλληλουχία
- αλφάβητο
- άλφα
- δεδομένα
- δένδρο



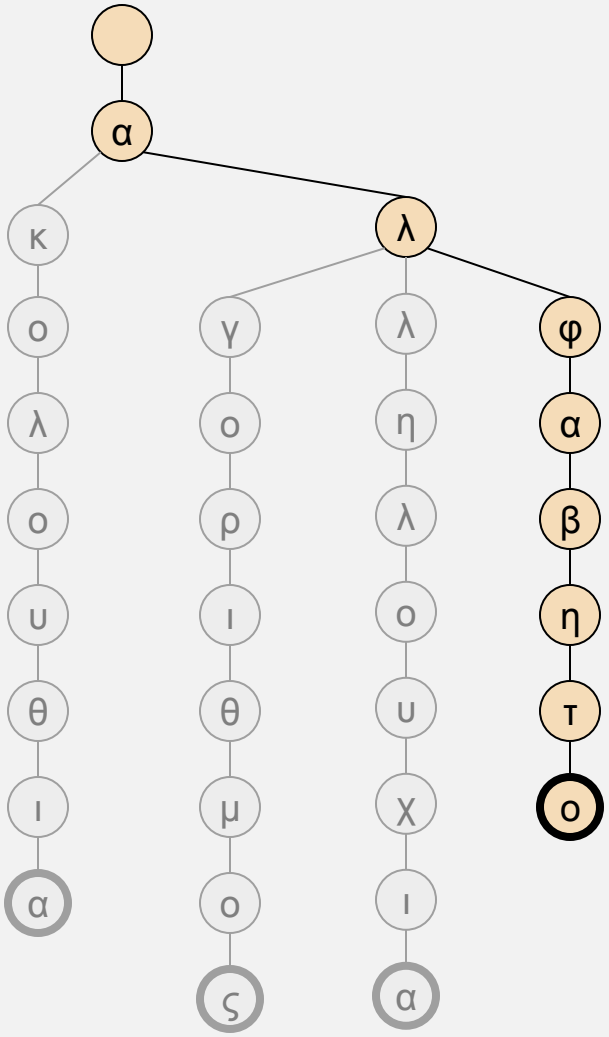
Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι

Tries για αποθήκευση αλφαριθμητικών

Παράδειγμα:

Trie για τις λέξεις

- ακολουθία
- αλγόριθμος
- αλληλουχία
- ⇨ αλφάβητο
- άλφα
- δεδομένα
- δένδρο



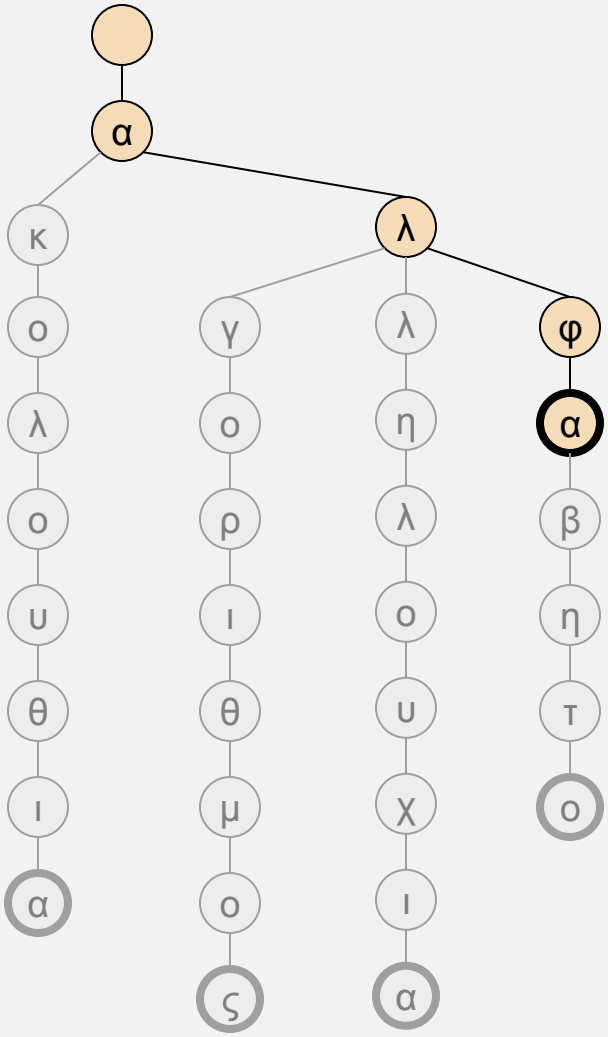
Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι

Tries για αποθήκευση αλφαριθμητικών

Παράδειγμα:

Trie για τις λέξεις

- ακολουθία
- αλγόριθμος
- αλληλουχία
- αλφάβητο
- ⇨ άλφα
- δεδομένα
- δένδρο



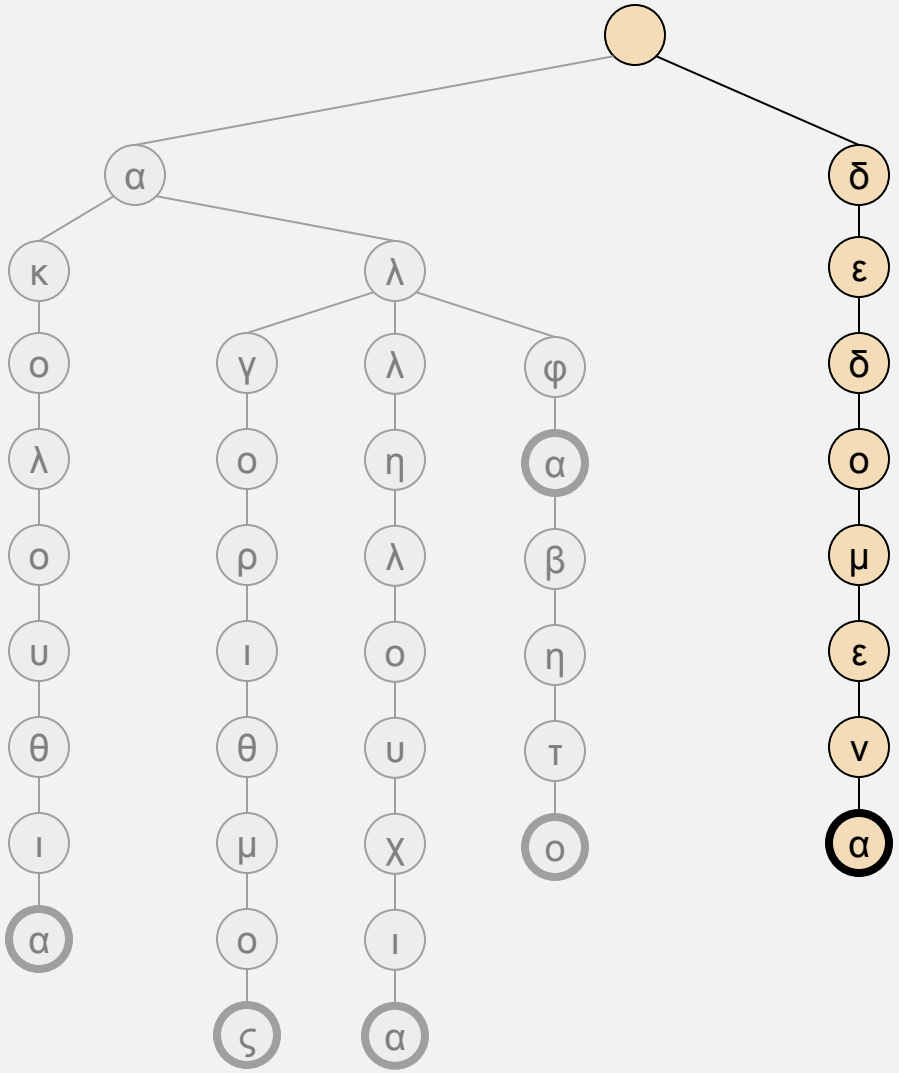
Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι

Tries για αποθήκευση αλφαριθμητικών

Παράδειγμα:

Trie για τις λέξεις

- ακολουθία
- αλγόριθμος
- αλληλουχία
- αλφάβητο
- άλφα
- ⇨ δεδομένα
- δένδρο



Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι

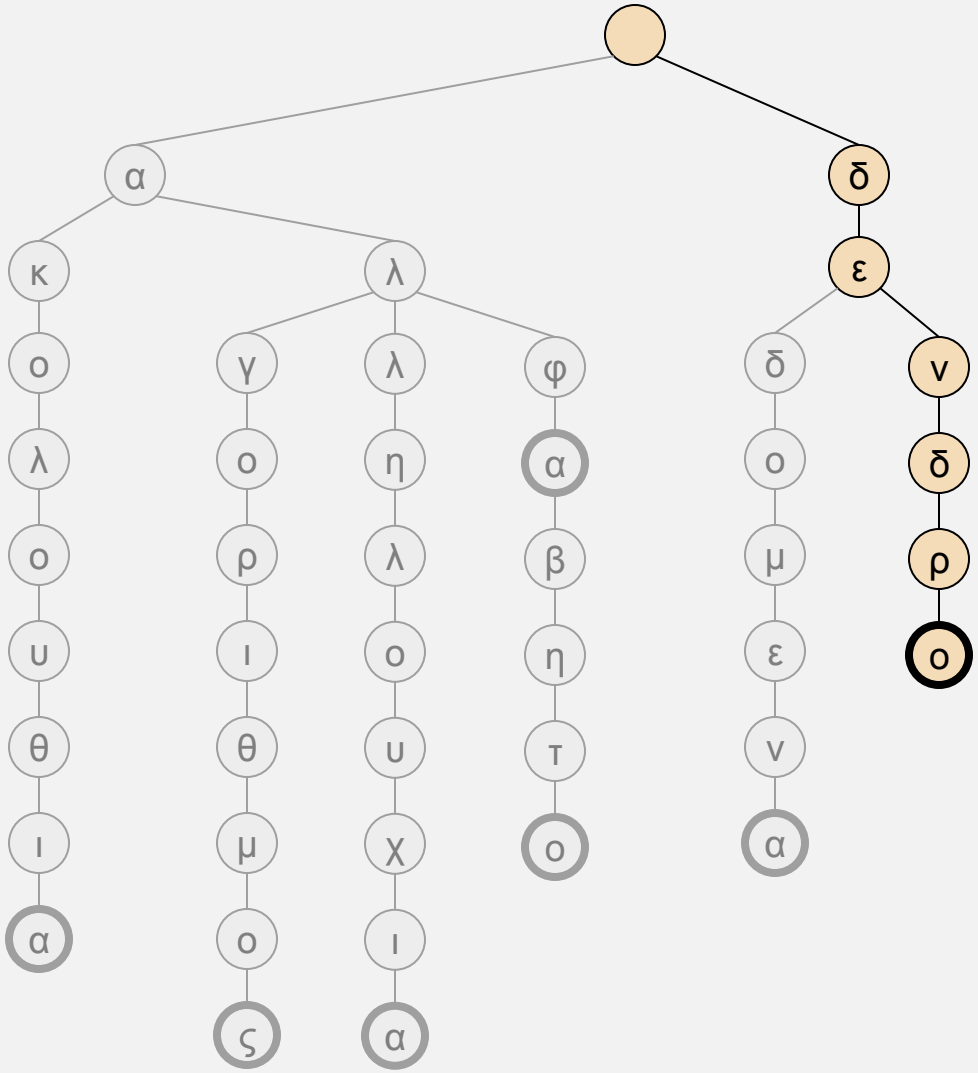
Tries για αποθήκευση αλφαριθμητικών

Παράδειγμα:

Trie για τις λέξεις

- ακολουθία
- αλγόριθμος
- αλληλουχία
- αλφάβητο
- άλφα
- δεδομένα
- ⇨ δένδρο

Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι 



Tries για αποθήκευση αλφαριθμητικών

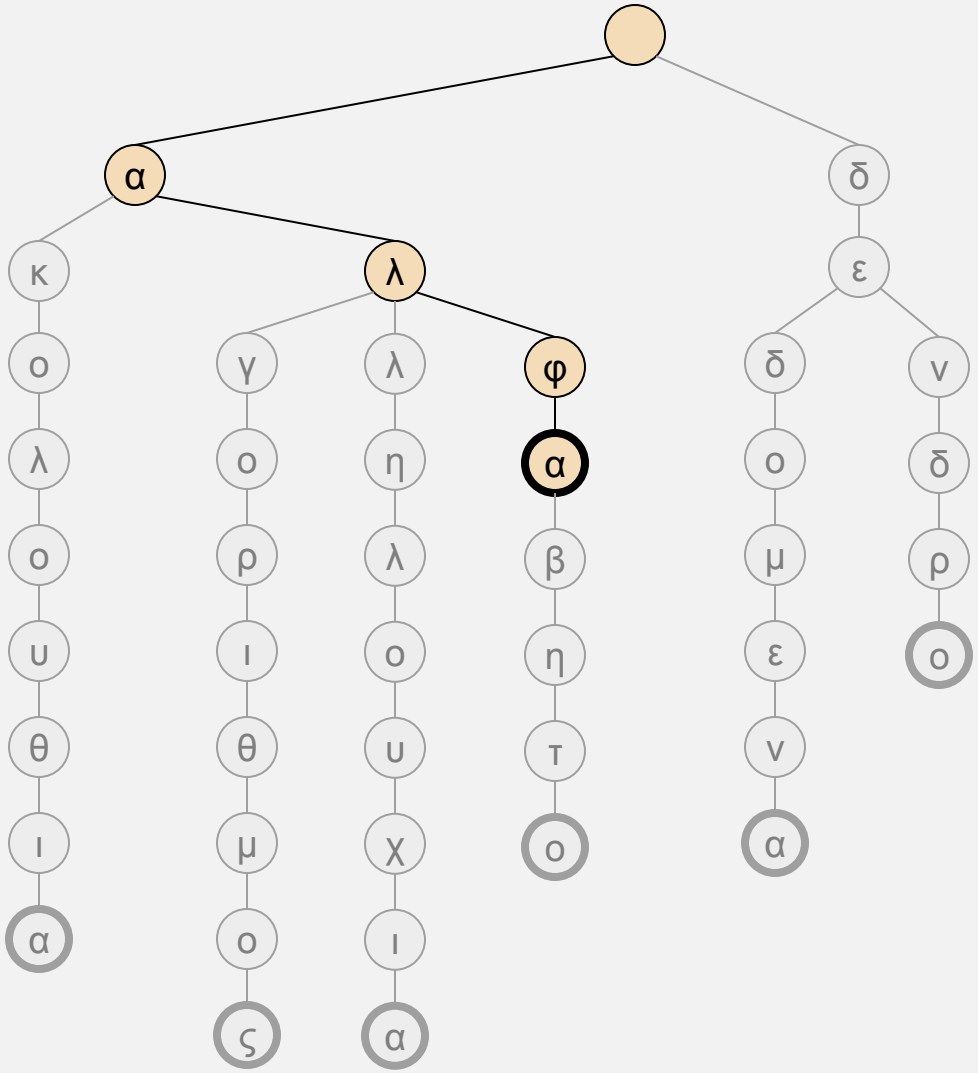
Διασύνδεση trie

`StringTrie()`	κατασκευή δομής trie
`void insert(String s)`	εισαγωγή αλφαριθμητικού s
`boolean contains(String s)`	έλεγχος αν το trie περιέχει το αλφαριθμητικό s
`Void delete(String s)`	διαγραφή του αλφαριθμητικού s
`String longestPrefixOf(String s)`	επιστρέφει το μεγαλύτερο αλφαριθμητικό που είναι πρόθεμα του s
`Iterable<String> keysWithPrefix(String s)`	επιστρέφει όλα τα αλφαριθμητικά που αποτελούν πρόθεμα του s
`Iterable<String> keysThatMatch(String s)`	επιστρέφει όλα τα αλφαριθμητικά που ταιριάζουν με το s (το s περιέχει ειδικούς χαρακτήρες οι οποίοι ταιριάζουν οποιοδήποτε χαρακτήρα)
`int size()`	αριθμός αντικειμένων στη δομή λεξικού
`Iterable<String> keys()`	επιστρέφει όλα τα αλφαριθμητικά του trie

Tries για αποθήκευση αλφαριθμητικών

```
contains("αλφα") = true
```

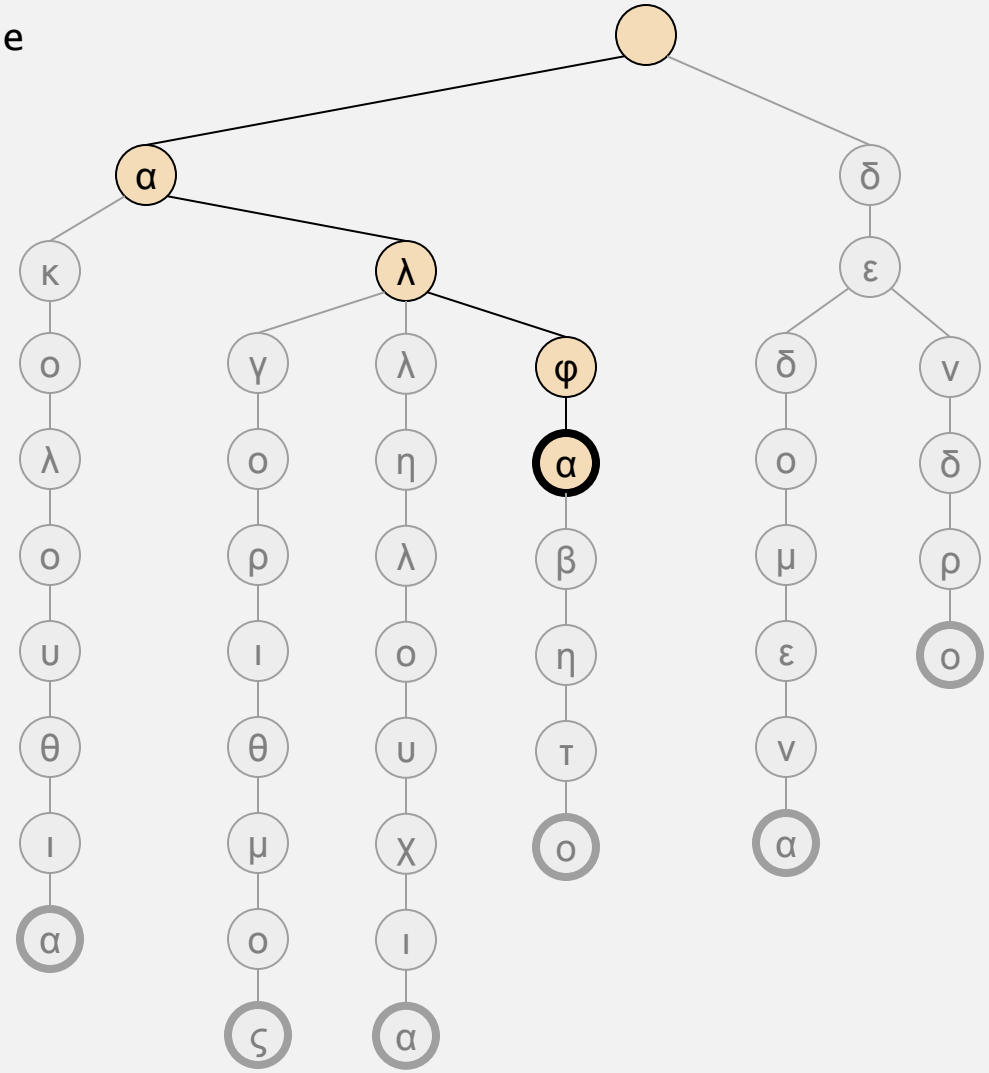
Η αναζήτηση καταλήγει σε κόμβο με επισήμανση



Tries για αποθήκευση αλφαριθμητικών

```
contains("αλφαριθμητικο") = false
```

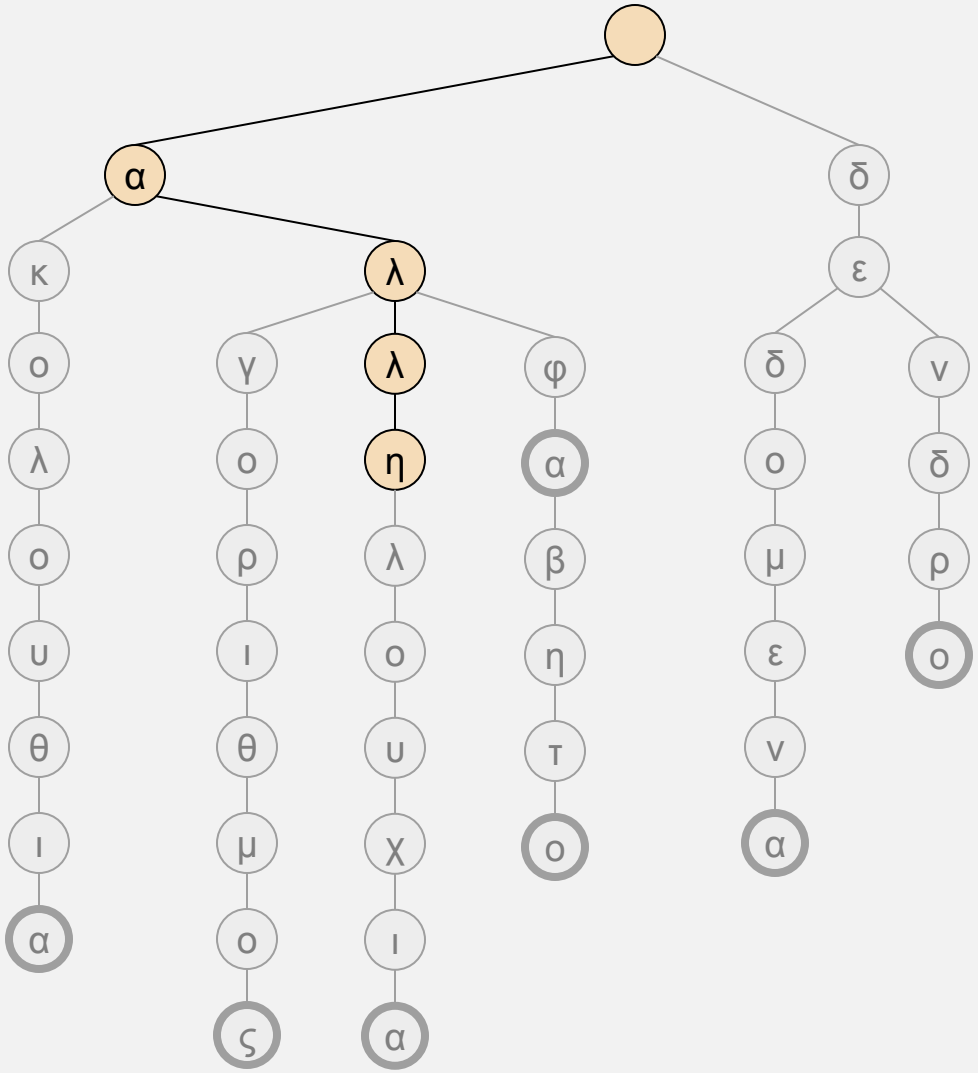
Η αναζήτηση καταλήγει σε κενό σύνδεσμο



Tries για αποθήκευση αλφαριθμητικών

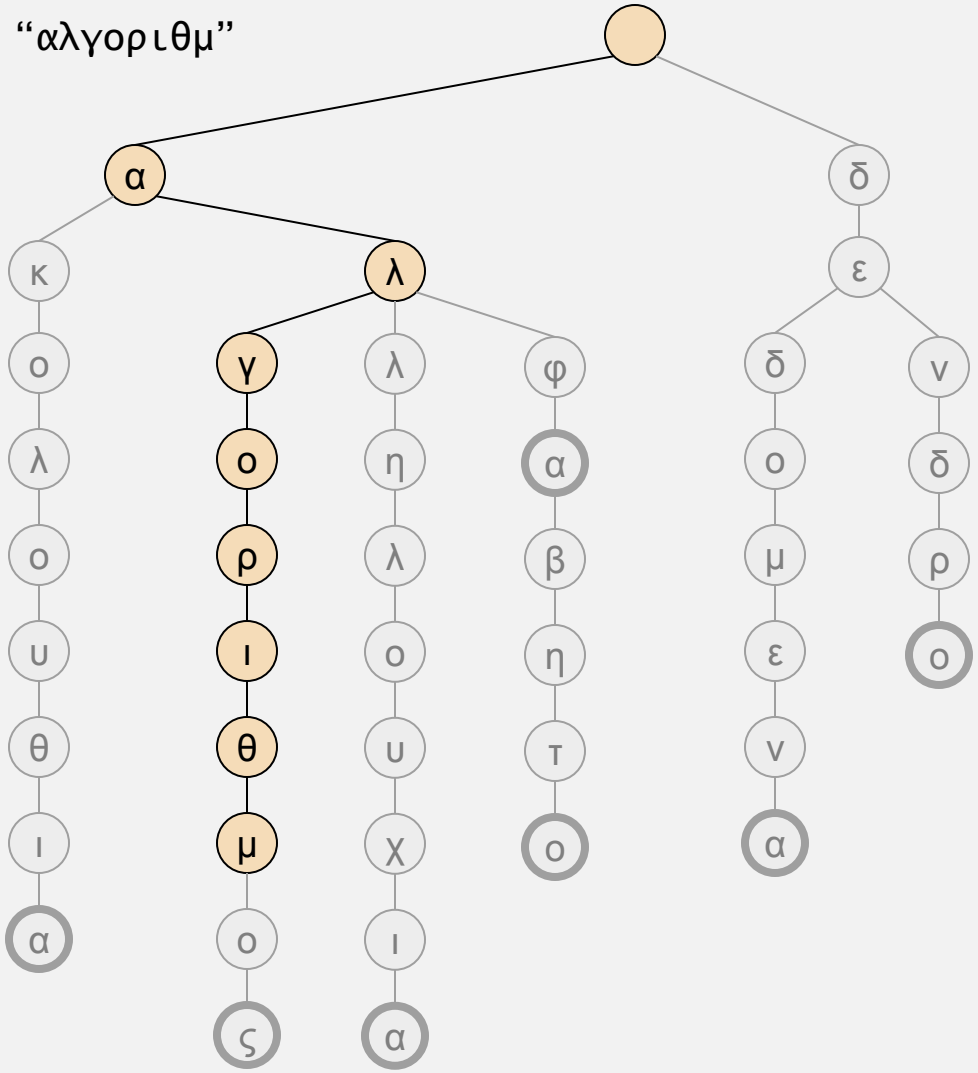
```
contains("αλλη") = false
```

Η αναζήτηση καταλήγει σε κόμβο χωρίς επισήμανση



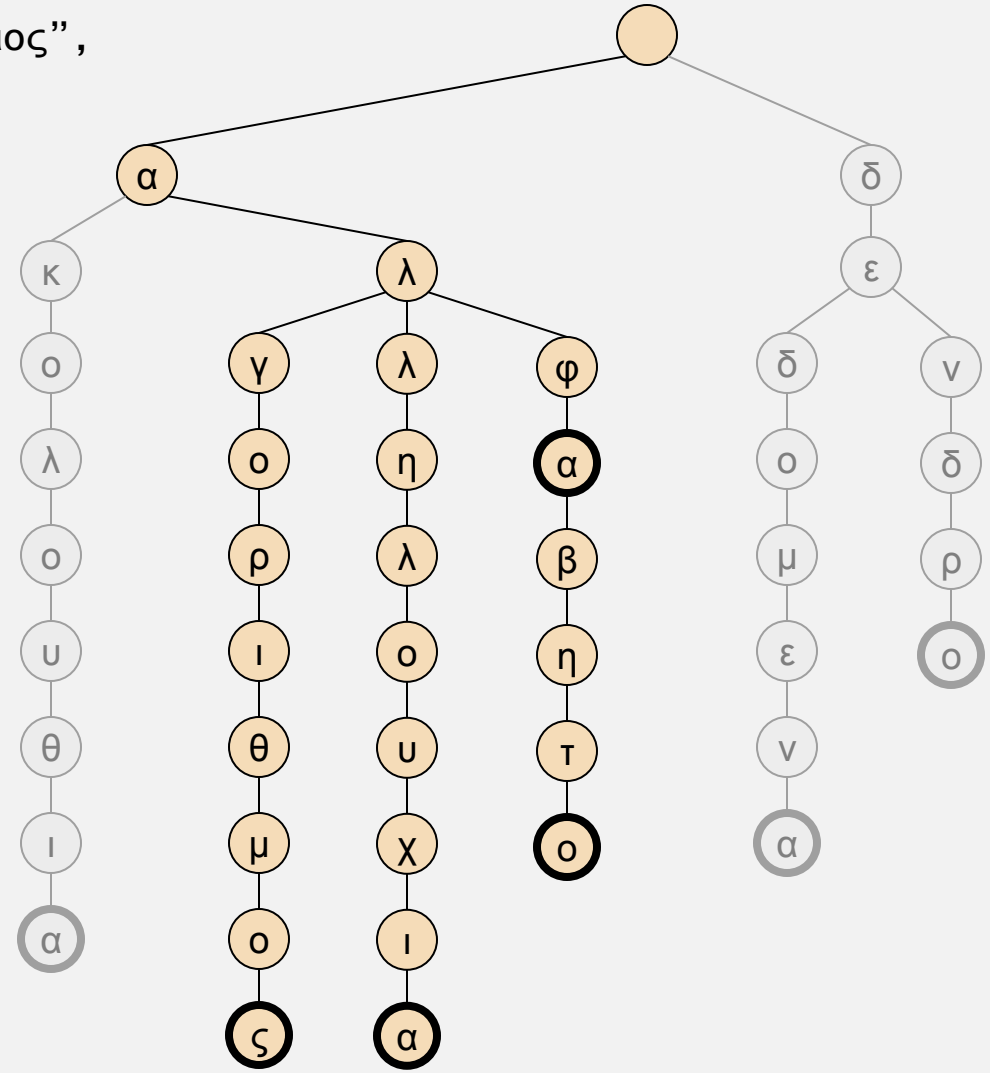
Tries για αποθήκευση αλφαριθμητικών

```
longestPrefixOf("αλγοριθμικη") = "αλγοριθμ"
```



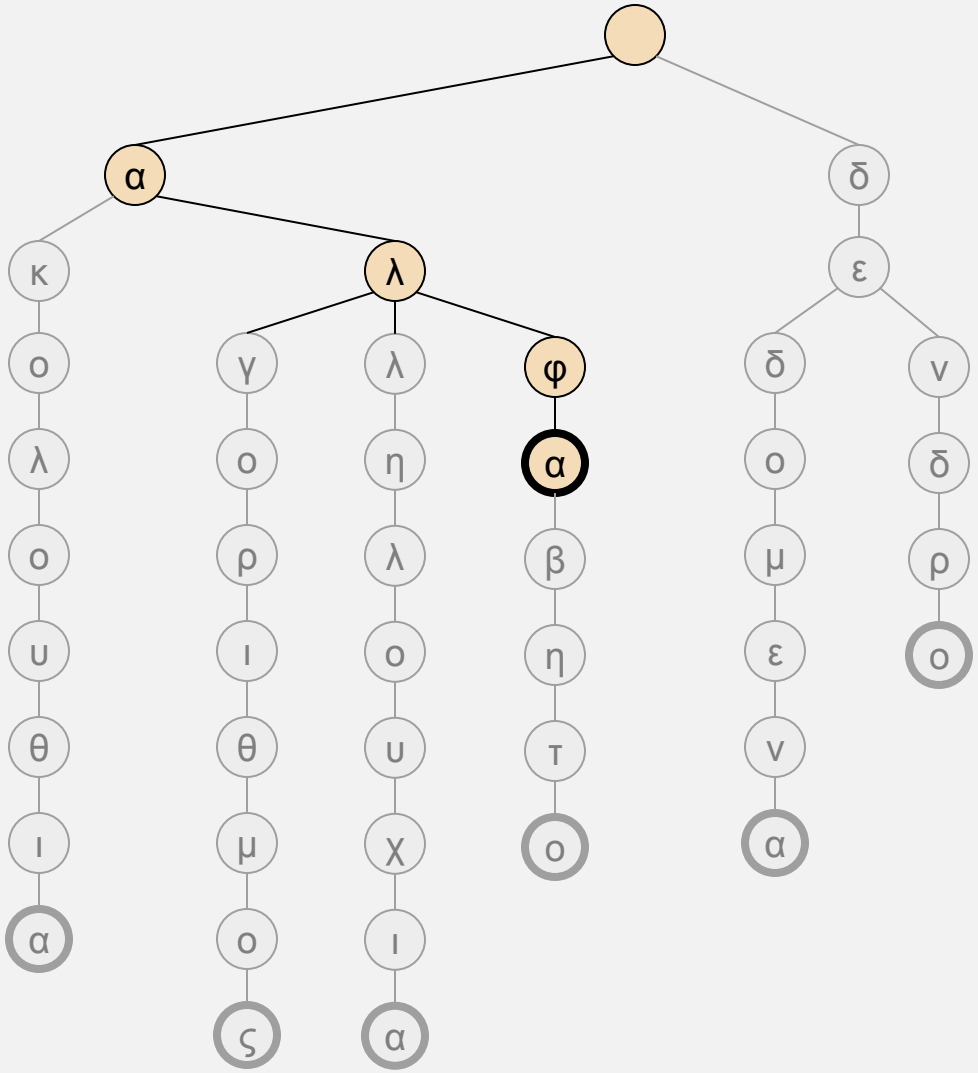
Tries για αποθήκευση αλφαριθμητικών

keysWithPrefix("αλ") = "αλγοριθμος", "αλληλουχία", "αλφα", "αλφαβητο"



Tries για αποθήκευση αλφαριθμητικών

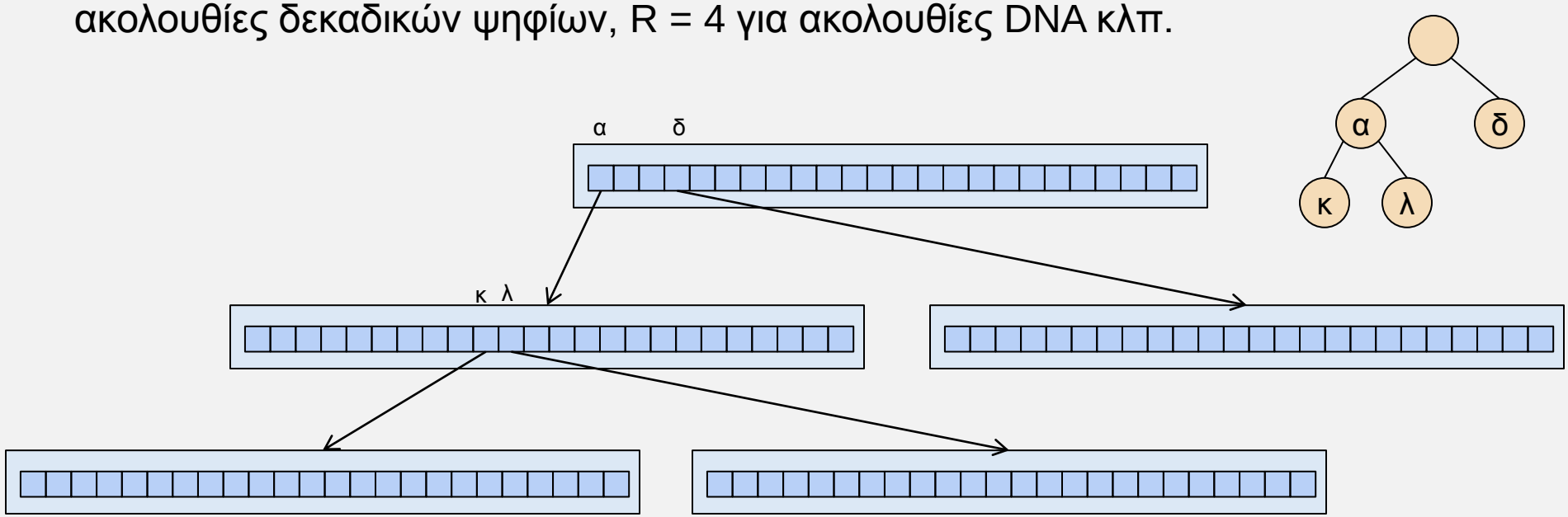
```
keysThatMatch("α??α") = "αλφα"
```



Tries για αποθήκευση αλφαριθμητικών

Αναπαράσταση :

Κάθε κόμβος αποθηκεύει ένα πίνακα συνδέσμων σε R παιδιά, όπου R το μέγεθος του αλφάβητου. Π.χ. R = 256 για χαρακτήρες 8 bit, R = 10 για ακολουθίες δεκαδικών ψηφίων, R = 4 για ακολουθίες DNA κλπ.



ελληνικοί χαρακτήρες : R = 24

Ο χαρακτήρας που αντιστοιχεί σε κάθε κόμβο αποθηκεύεται έμμεσα: αντιστοιχεί στο σύνδεσμο που ακολουθήθηκε από τον γονέα του κόμβου.

Tries για αποθήκευση αλφαριθμητικών

```
public class StringTrie {
    private static int R = 256;  // πλήθος διαφορετικών χαρακτήρων
    private Node root;
    private static class Node {
         private boolean mark;  // bit επισήμανσης κόμβου (true αν είναι τέλος λέξης)
         private Node[] next = new Node [R];  // σύνδεσμοι σε R παιδιά
    }

    public boolean contains(String s) {
        Node x = contains(root, s, 0);
        if (x == null) { return false; } else { return x.mark; }
    }
    private Node contains(Node x, String s, int d) { // αναζήτηση στο υποδένδρο του x
       if (x == null) return null;
       if (d == s.length()) return x;  // τέλος αναζητούμενης λέξης
       char c = s.charAt(d);  // επόμενος χαρακτήρας
       return contains(x.next[c], s, d+1);
    }

    public void insert(String s) { root = insert(root, s, 0); }
    private Node insert(Node x, String s, int d) { // εισαγωγή στο υποδένδρο του x
       if (x == null) x = new Node();
       if (d == s.length()) { x.mark = true; return x; }  // τέλος λέξης
       char c = s.charAt(d); // επόμενος χαρακτήρας
       x.next[c] = insert(x.next[c], s, d+1);
       return x;
    }
}
```

Tries για αποθήκευση αλφαριθμητικών

Ιδιότητες

- Η μορφή του trie είναι ανεξάρτητη από τη σειρά εισαγωγής των στοιχείων. Κάθε δεδομένο σύνολο διακριτών στοιχείων δημιουργεί ένα μοναδικό trie.

- Η αναζήτηση ή εισαγωγή ενός στοιχείου μήκους l χαρακτήρων απαιτεί χρόνο $O(l)$ στη χειρότερη περίπτωση.

- Η ανεπιτυχής αναζήτηση ενός τυχαίου στοιχείου σε trie κατασκευασμένο από n τυχαίες (διακεκριμένες) ακολουθίες από αλφάβητο R χαρακτήρων, απαιτεί κατά μέσο όρο χρόνο $O(\log_R n)$.

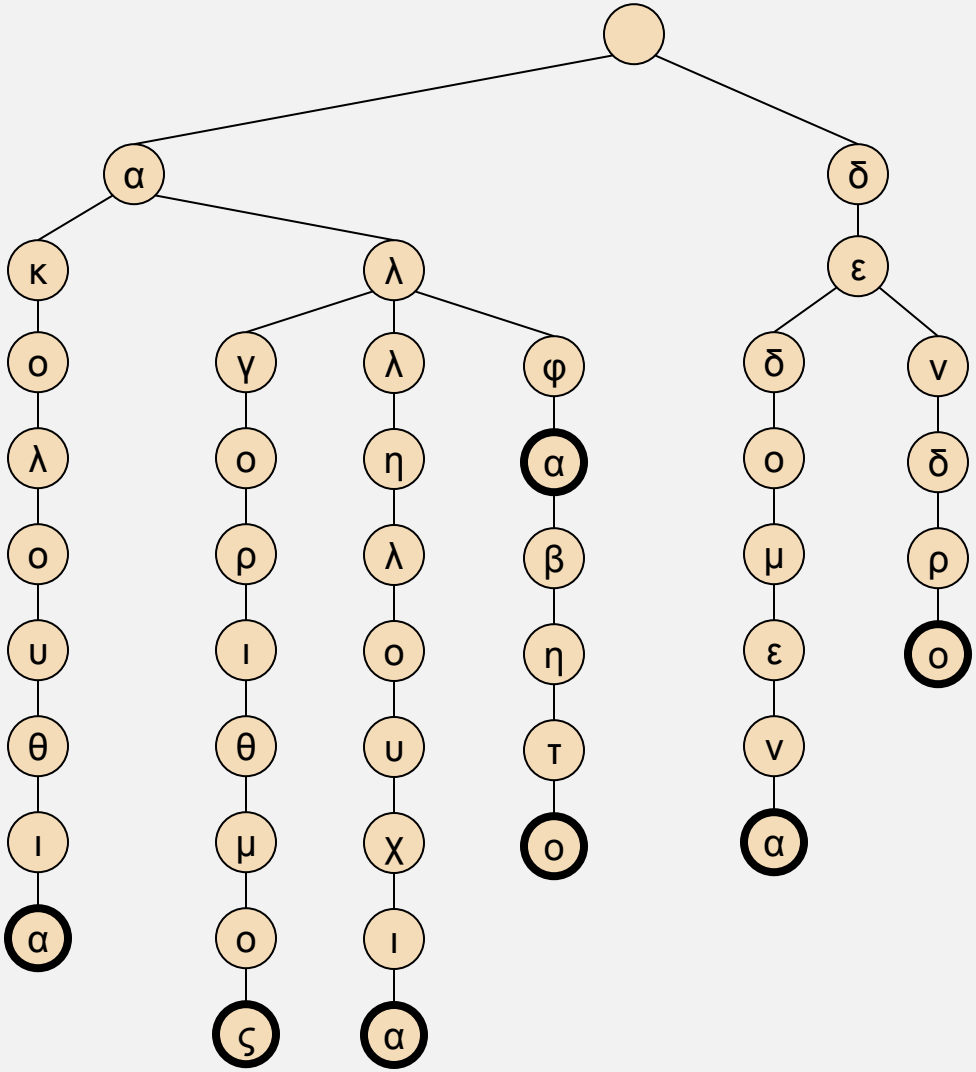
- Ένα trie κατασκευασμένο από n στοιχεία μέσου μήκους l από αλφάβητο R χαρακτήρων έχει πλήθος συνδέσμων μεταξύ $R \cdot n$ και $R \cdot n \cdot l$.

Tries για αποθήκευση αλφαριθμητικών

Διαγραφή λέξης :

Βρίσκουμε τον κόμβο x που αντιστοιχεί στο τελευταίο γράμμα της λέξης και σβήνουμε την επισήμανση του.

Αν ο x δεν έχει απογόνους στο trie τότε τον διαγράφουμε και συνεχίζουμε την ίδια διαδικασία στους προγόνους του x μέχρι να φτάσουμε σε κόμβο με απογόνους ή με επισήμανση.

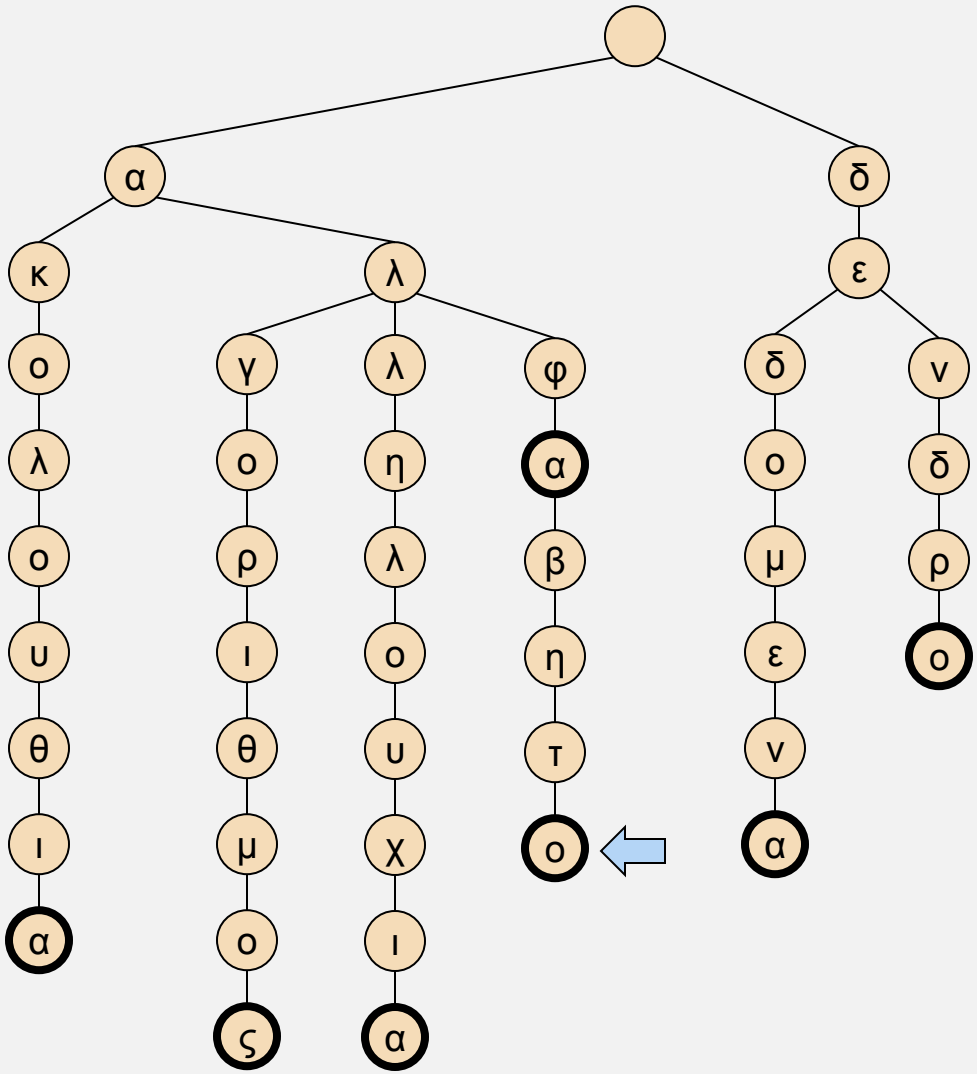


Tries για αποθήκευση αλφαριθμητικών

Διαγραφή λέξης :

Βρίσκουμε τον κόμβο x που αντιστοιχεί στο τελευταίο γράμμα της λέξης και σβήνουμε την επισήμανση του.

Αν ο x δεν έχει απογόνους στο trie τότε τον διαγράφουμε και συνεχίζουμε την ίδια διαδικασία στους προγόνους του x μέχρι να φτάσουμε σε κόμβο με απογόνους ή με επισήμανση.



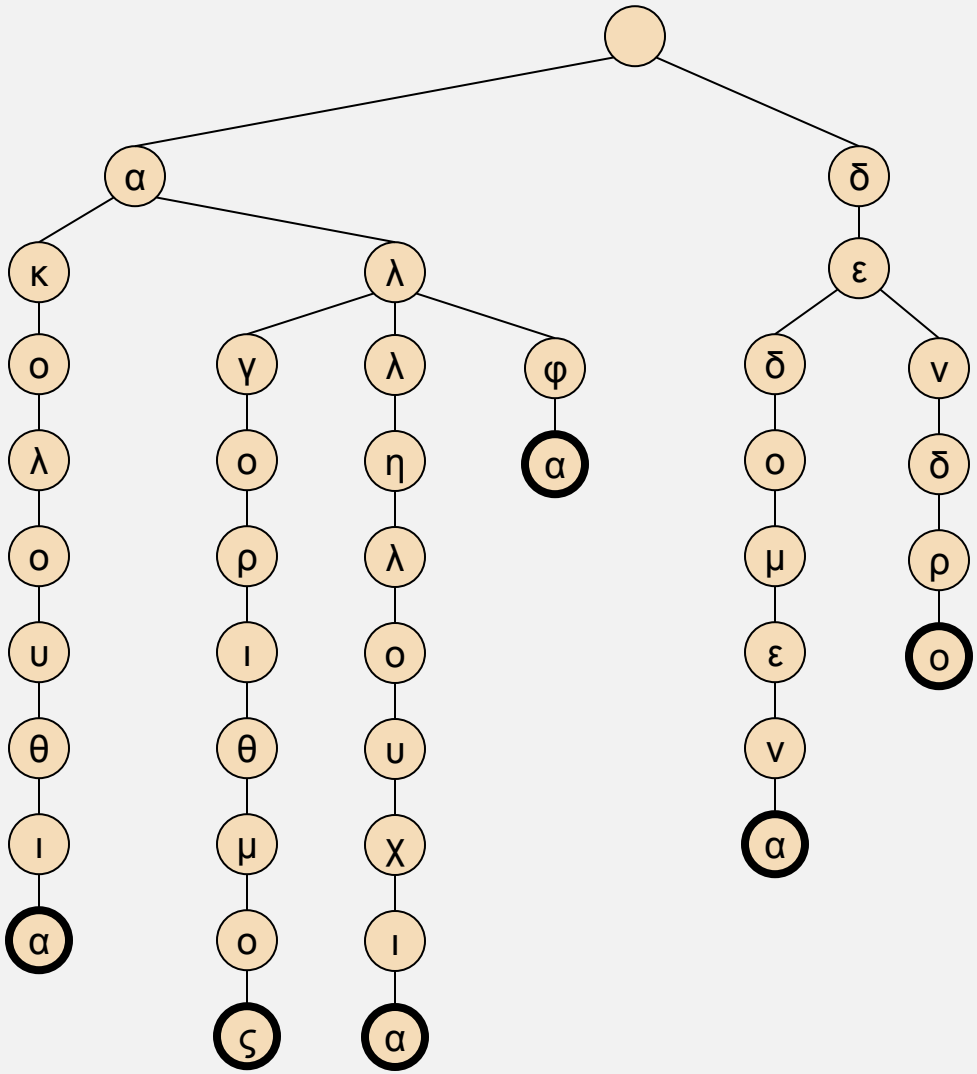
`delete("αλφαβητο")`

Tries για αποθήκευση αλφαριθμητικών

Διαγραφή λέξης :

Βρίσκουμε τον κόμβο x που αντιστοιχεί στο τελευταίο γράμμα της λέξης και σβήνουμε την επισήμανση του.

Αν ο x δεν έχει απογόνους στο trie τότε τον διαγράφουμε και συνεχίζουμε την ίδια διαδικασία στους προγόνους του x μέχρι να φτάσουμε σε κόμβο με απογόνους ή με επισήμανση.



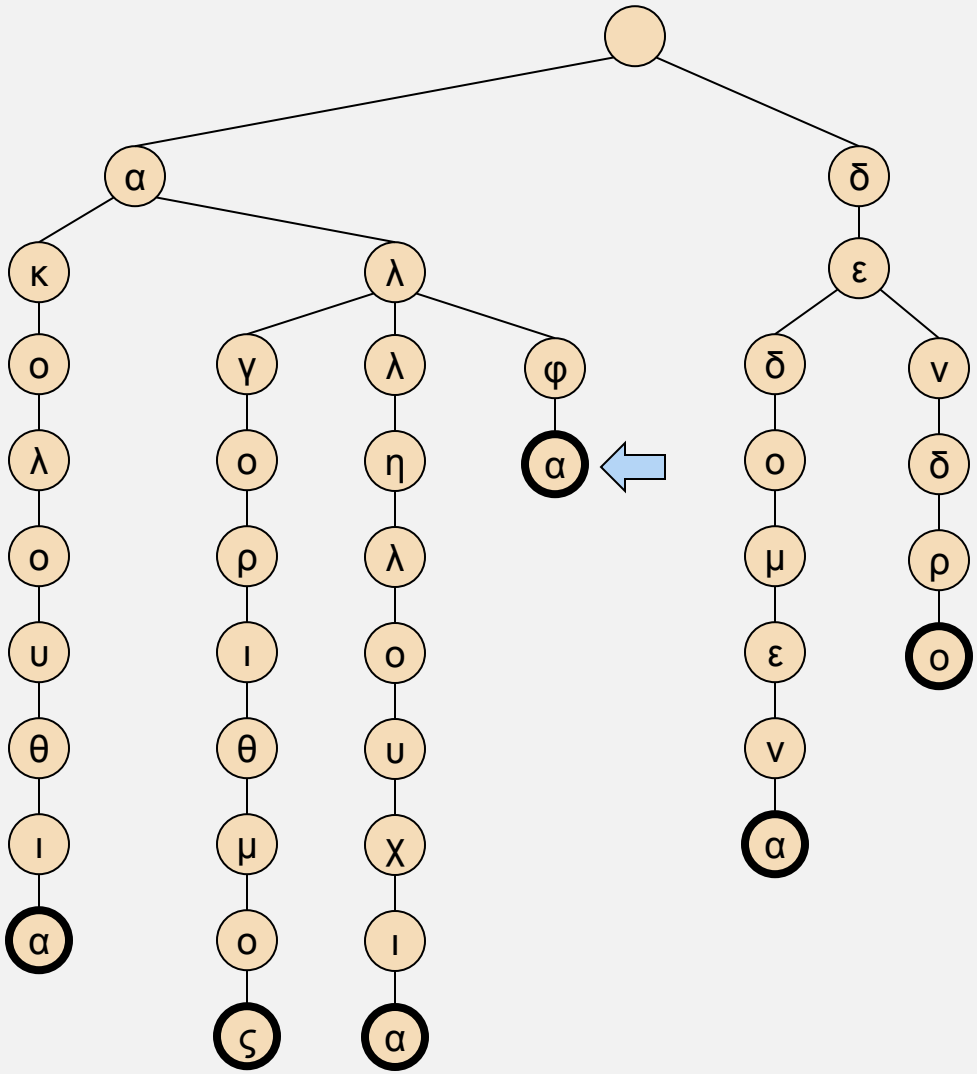
`delete("αλφαβητο")`

Tries για αποθήκευση αλφαριθμητικών

Διαγραφή λέξης :

Βρίσκουμε τον κόμβο x που αντιστοιχεί στο τελευταίο γράμμα της λέξης και σβήνουμε την επισήμανση του.

Αν ο x δεν έχει απογόνους στο trie τότε τον διαγράφουμε και συνεχίζουμε την ίδια διαδικασία στους προγόνους του x μέχρι να φτάσουμε σε κόμβο με απογόνους ή με επισήμανση.



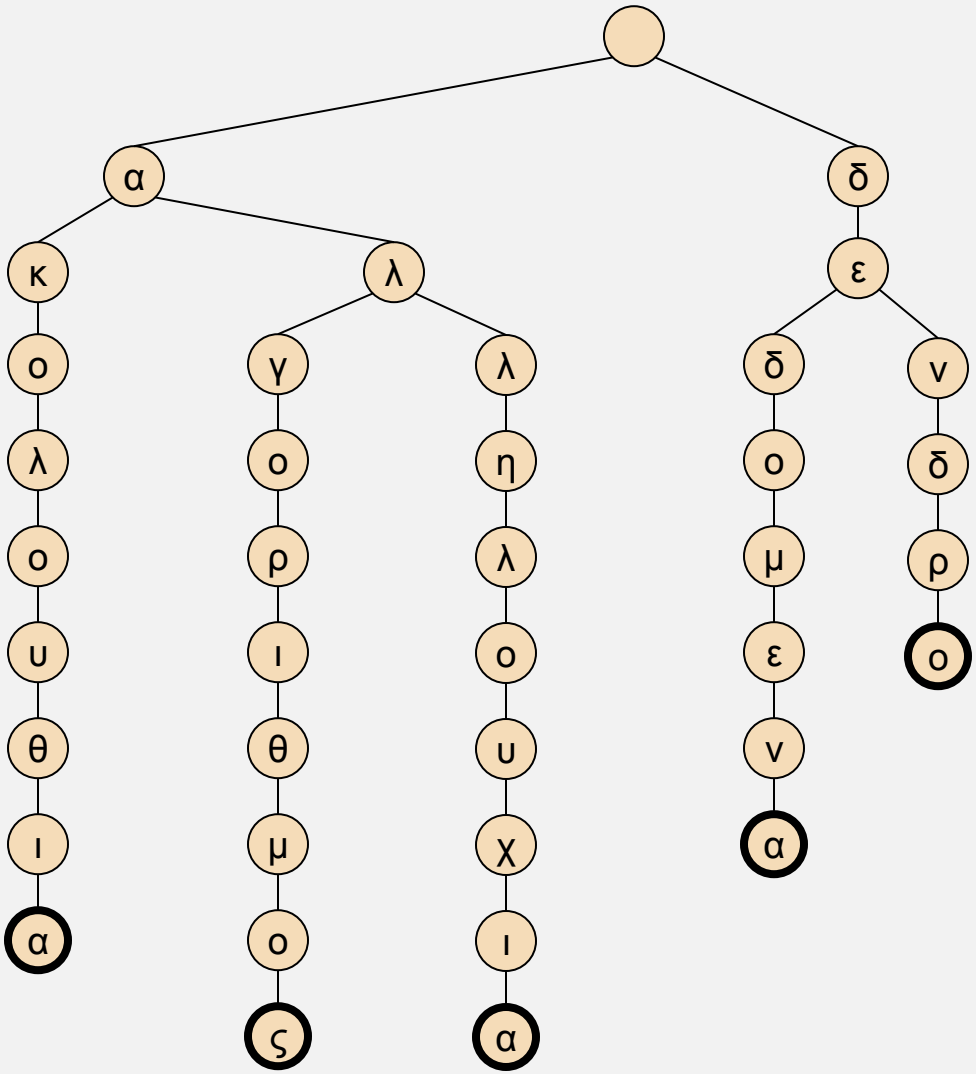
`delete("αλφα")`

Tries για αποθήκευση αλφαριθμητικών

Διαγραφή λέξης :

Βρίσκουμε τον κόμβο x που αντιστοιχεί στο τελευταίο γράμμα της λέξης και σβήνουμε την επισήμανση του.

Αν ο x δεν έχει απογόνους στο trie τότε τον διαγράφουμε και συνεχίζουμε την ίδια διαδικασία στους προγόνους του x μέχρι να φτάσουμε σε κόμβο με απογόνους ή με επισήμανση.



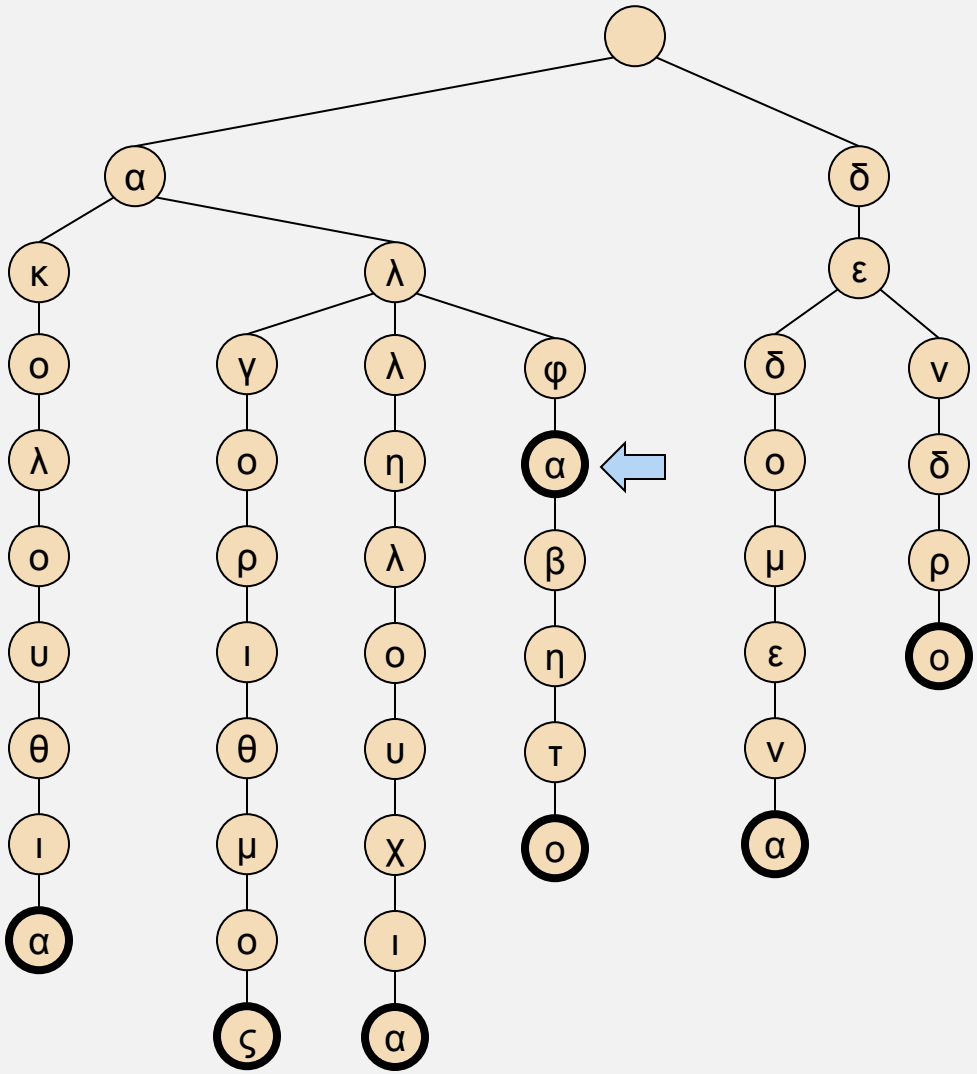
`delete("αλφα")`

Tries για αποθήκευση αλφαριθμητικών

Διαγραφή λέξης :

Βρίσκουμε τον κόμβο x που αντιστοιχεί στο τελευταίο γράμμα της λέξης και σβήνουμε την επισήμανση του.

Αν ο x δεν έχει απογόνους στο trie τότε τον διαγράφουμε και συνεχίζουμε την ίδια διαδικασία στους προγόνους του x μέχρι να φτάσουμε σε κόμβο με απογόνους ή με επισήμανση.



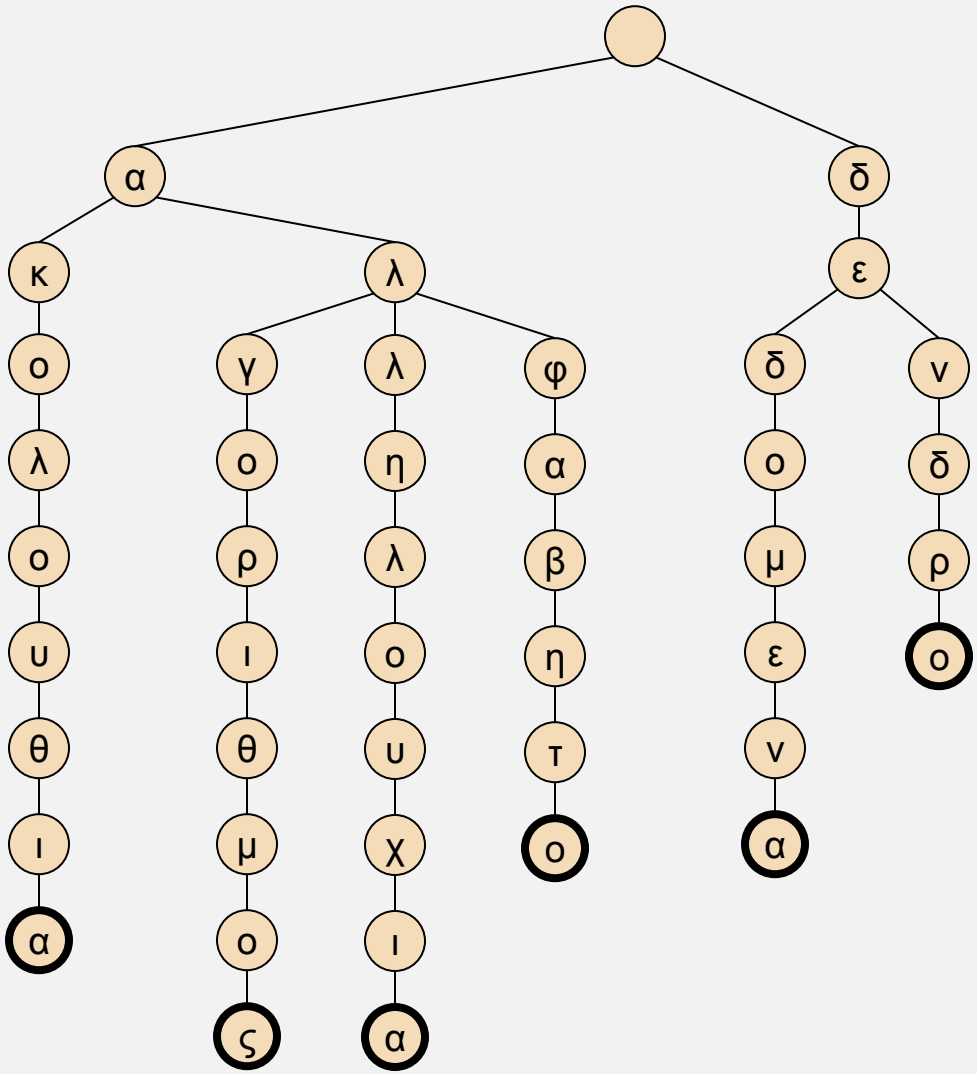
`delete("αλφα")`

Tries για αποθήκευση αλφαριθμητικών

Διαγραφή λέξης :

Βρίσκουμε τον κόμβο x που αντιστοιχεί στο τελευταίο γράμμα της λέξης και σβήνουμε την επισήμανση του.

Αν ο x δεν έχει απογόνους στο trie τότε τον διαγράφουμε και συνεχίζουμε την ίδια διαδικασία στους προγόνους του x μέχρι να φτάσουμε σε κόμβο με απογόνους ή με επισήμανση.



`delete("αλφα")`

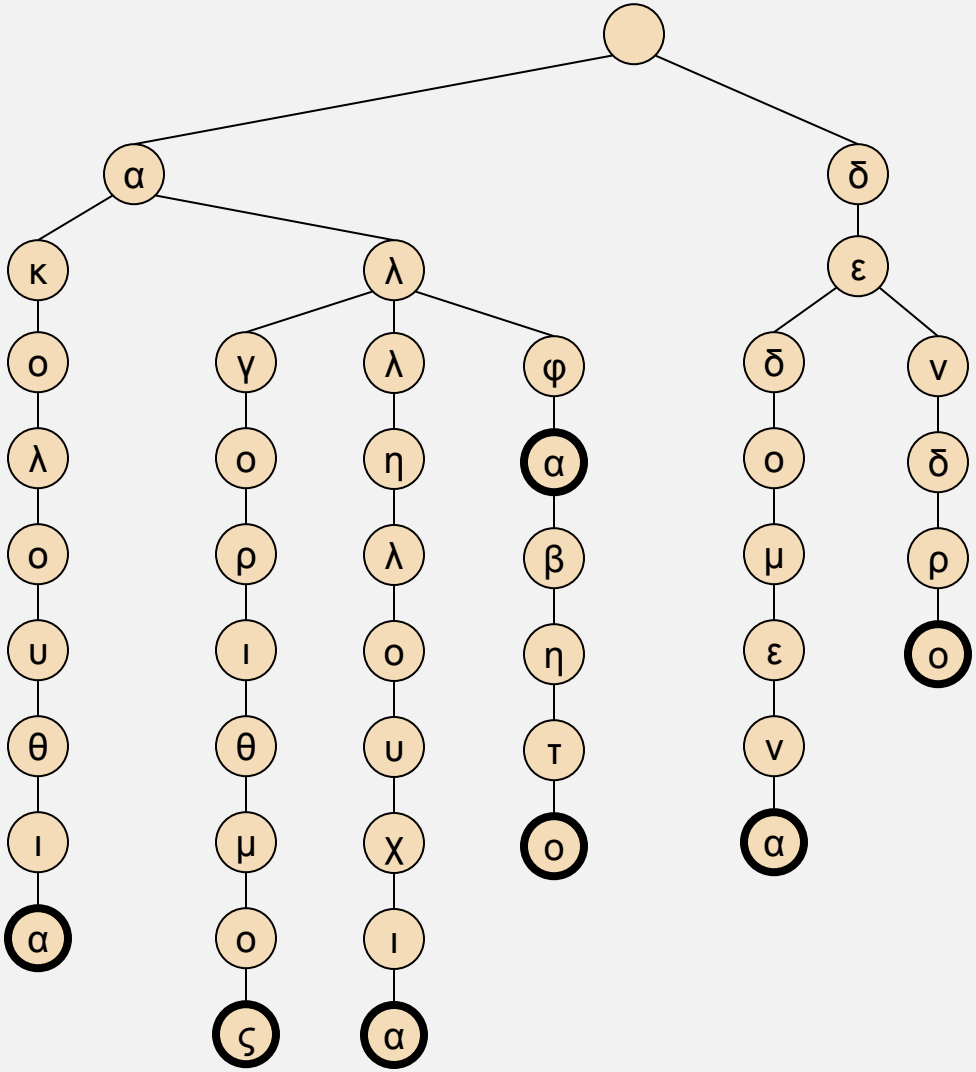
Tries για αποθήκευση αλφαριθμητικών

```
public class StringTrie {
    private static int R = 256;  // πλήθος διαφορετικών χαρακτήρων
    private Node root;
    private static class Node {
         private boolean mark;  // bit επισήμανσης κόμβου (true αν είναι τέλος λέξης)
         private Node[] next = new Node [R];  // σύνδεσμοι σε R παιδιά
    }
    …
    public void delete(String s) { root = delete(root, s, 0); }

    private Node delete(Node x, String s, int d) { // διαγραφή στο υποδένδρο του x
       if (x == null) return null;
       if (d == s.length())
          x.mark = false;  // τέλος λέξης: σβήνουμε την επισήμανση
       else
       {
          char c = s.charAt(d);  // επόμενος χαρακτήρας
          x.next[c] = delete(x.next[c], s, d+1);
       }

       if ( x.mark ) return x; // ο x έχει επισήμανση και επομένως δεν διαγράφεται

       for (char c=0; c<R; c++) {
          if ( x.next[c] != null )
             return x; // ο x έχει παιδιά και επομένως δεν διαγράφεται
       }
       return null;
    }
}
```

Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.



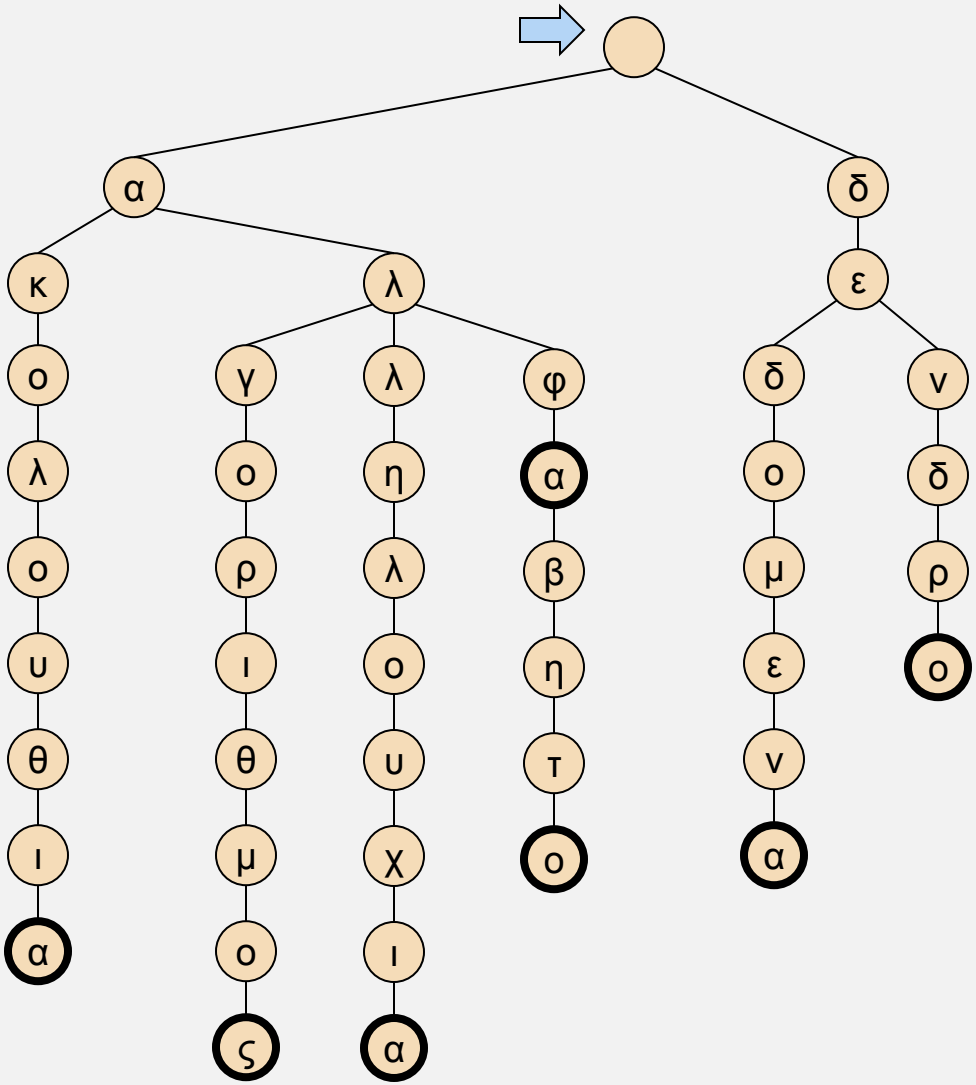
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “”
```

```
Queue q = ()
```



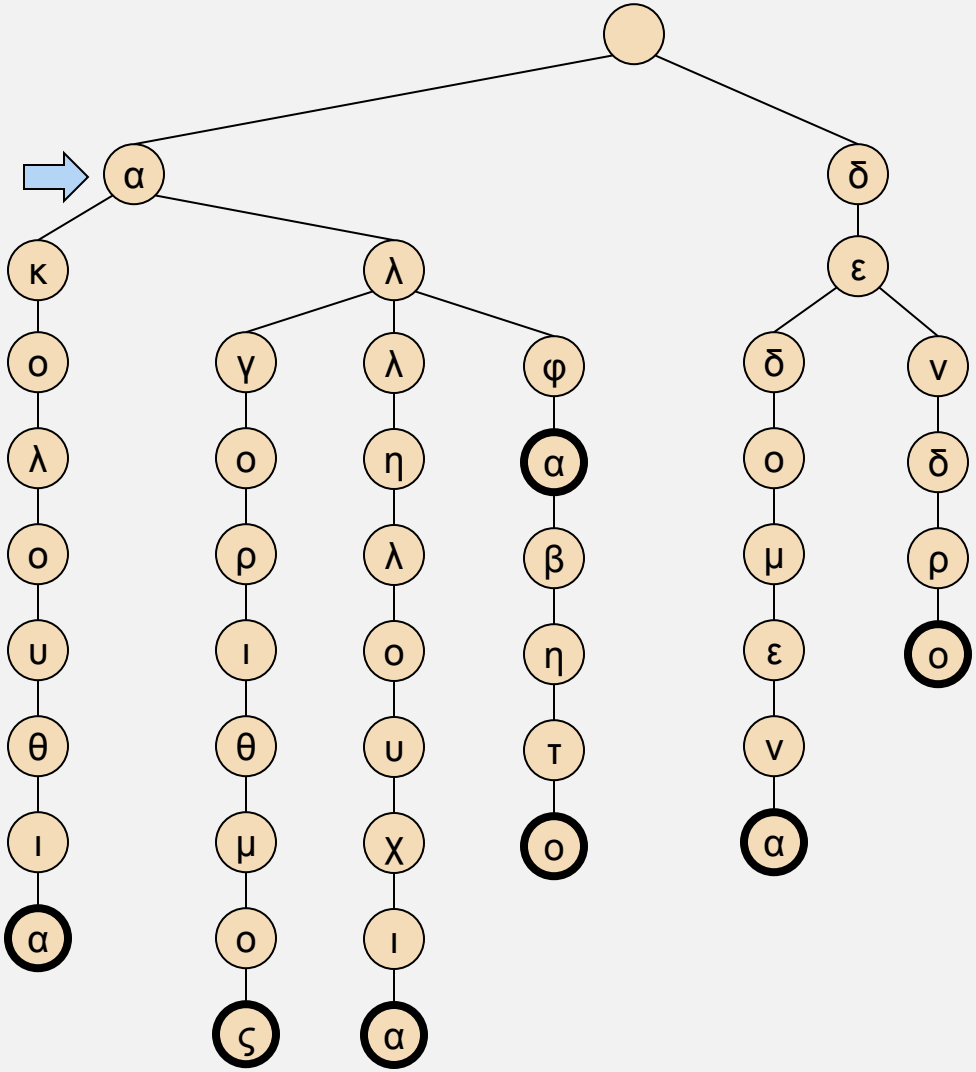
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “α”
```

```
Queue q = ()
```



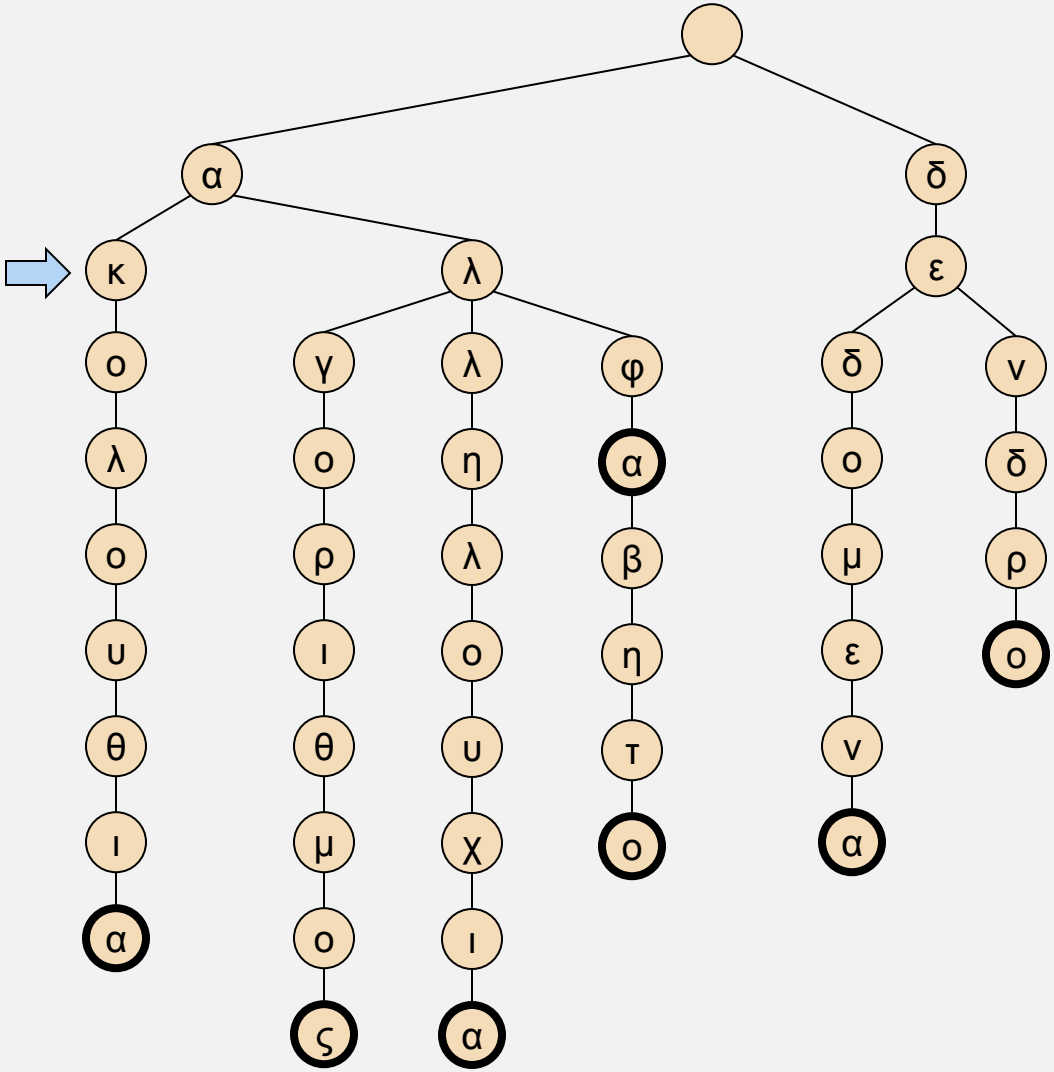
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “ακ”
```

```
Queue q = ()
```



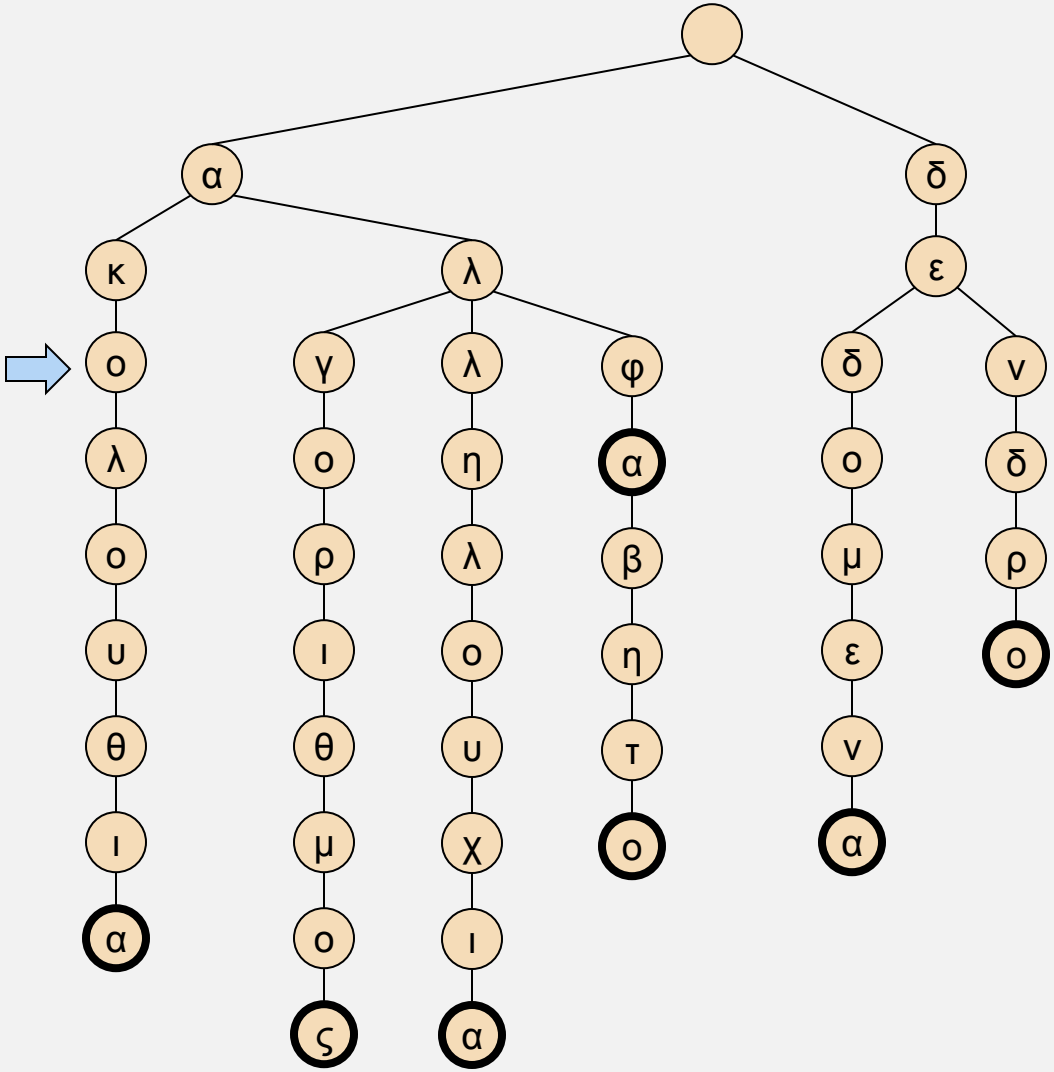
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “ακο”
```

```
Queue q = ()
```



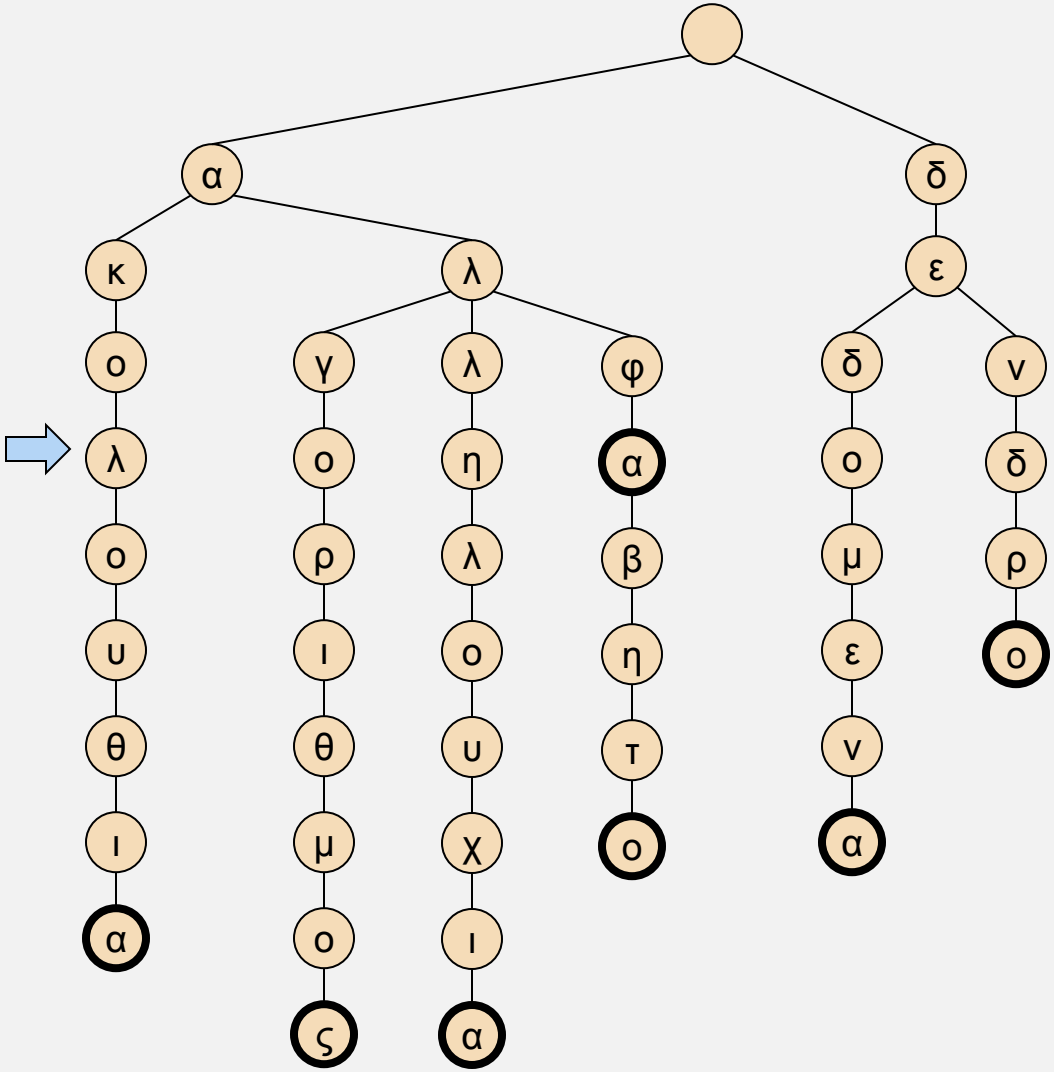
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “ακολ”
```

```
Queue q = ()
```



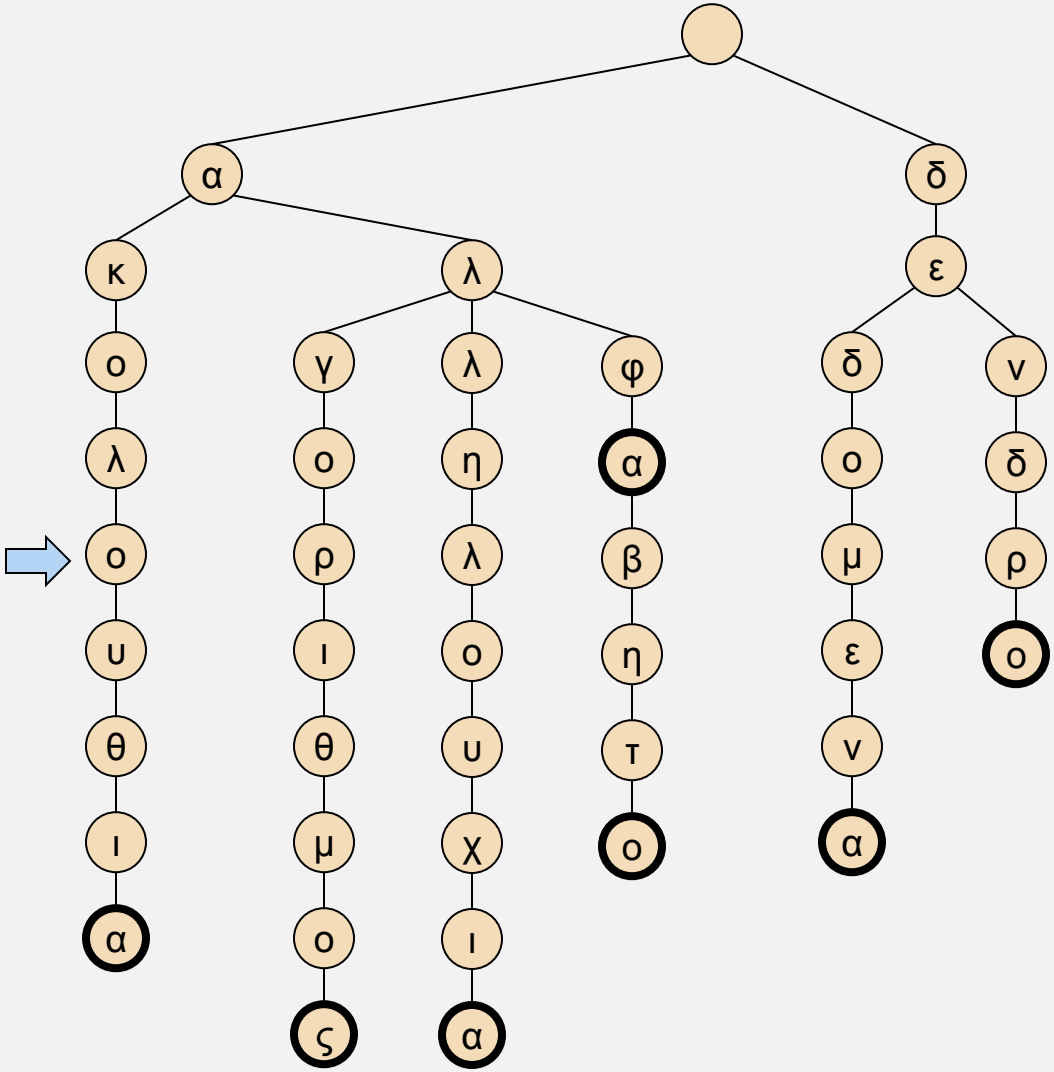
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = "ακολο"
```

```
Queue q = ()
```



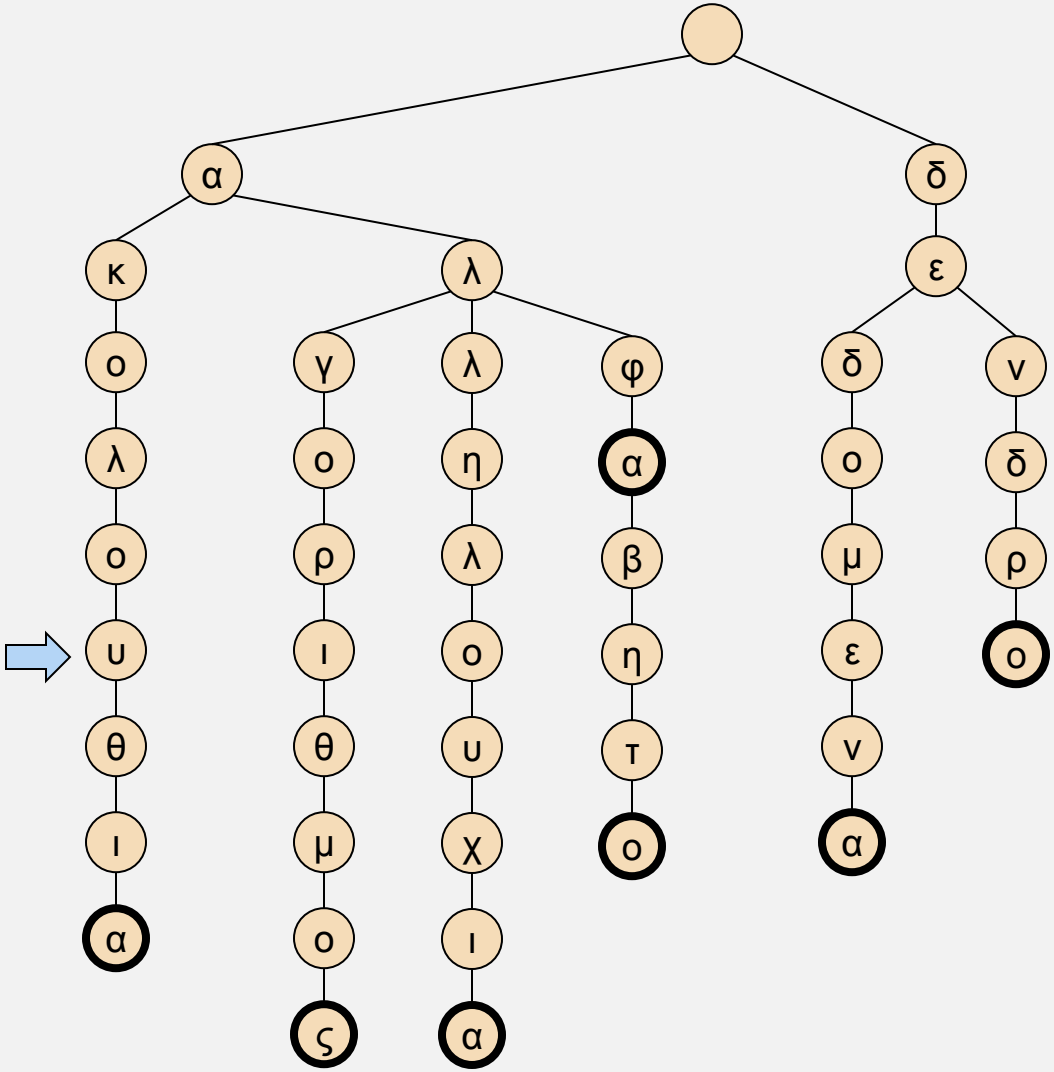
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “ακολου”
```

```
Queue q = ()
```



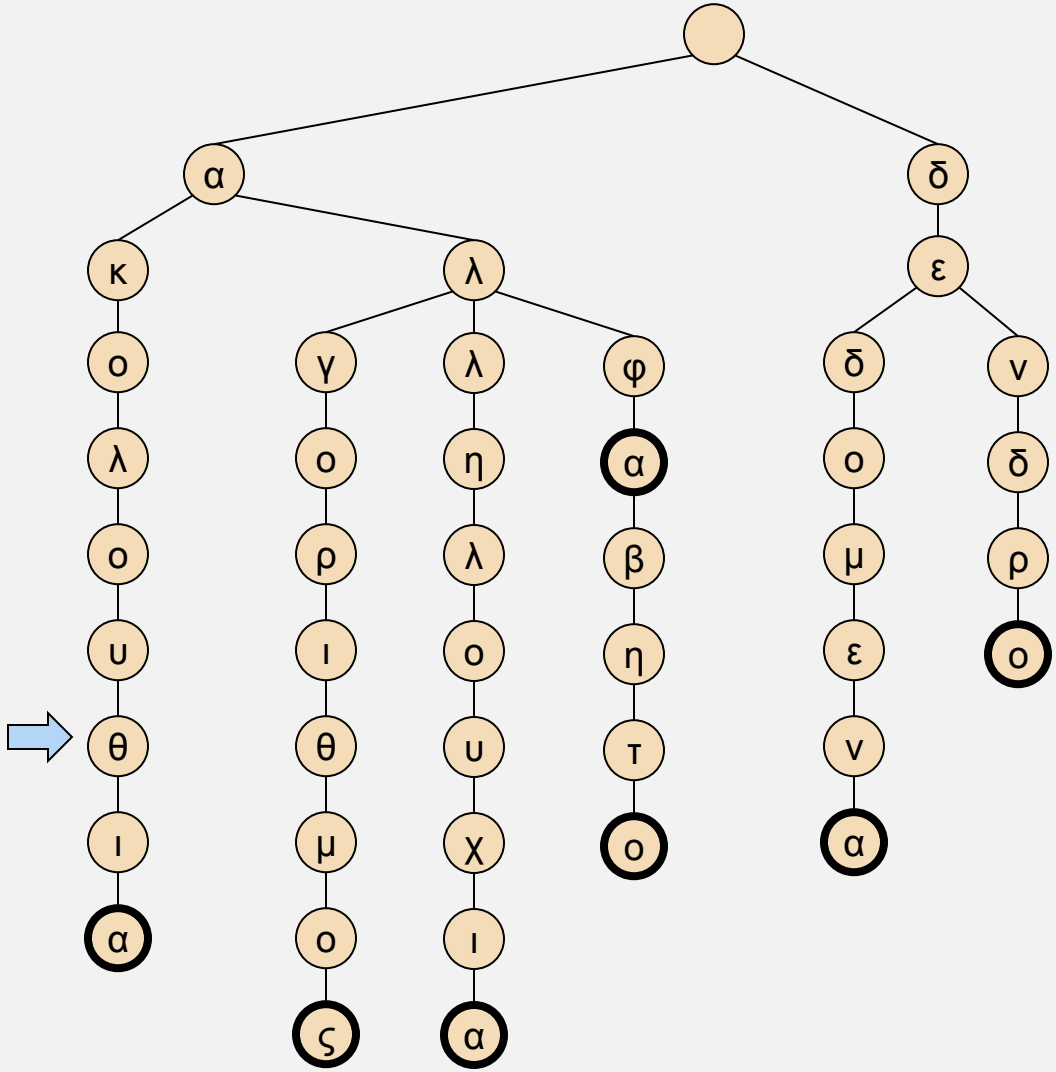
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “ακολουθ”
```

```
Queue q = ()
```



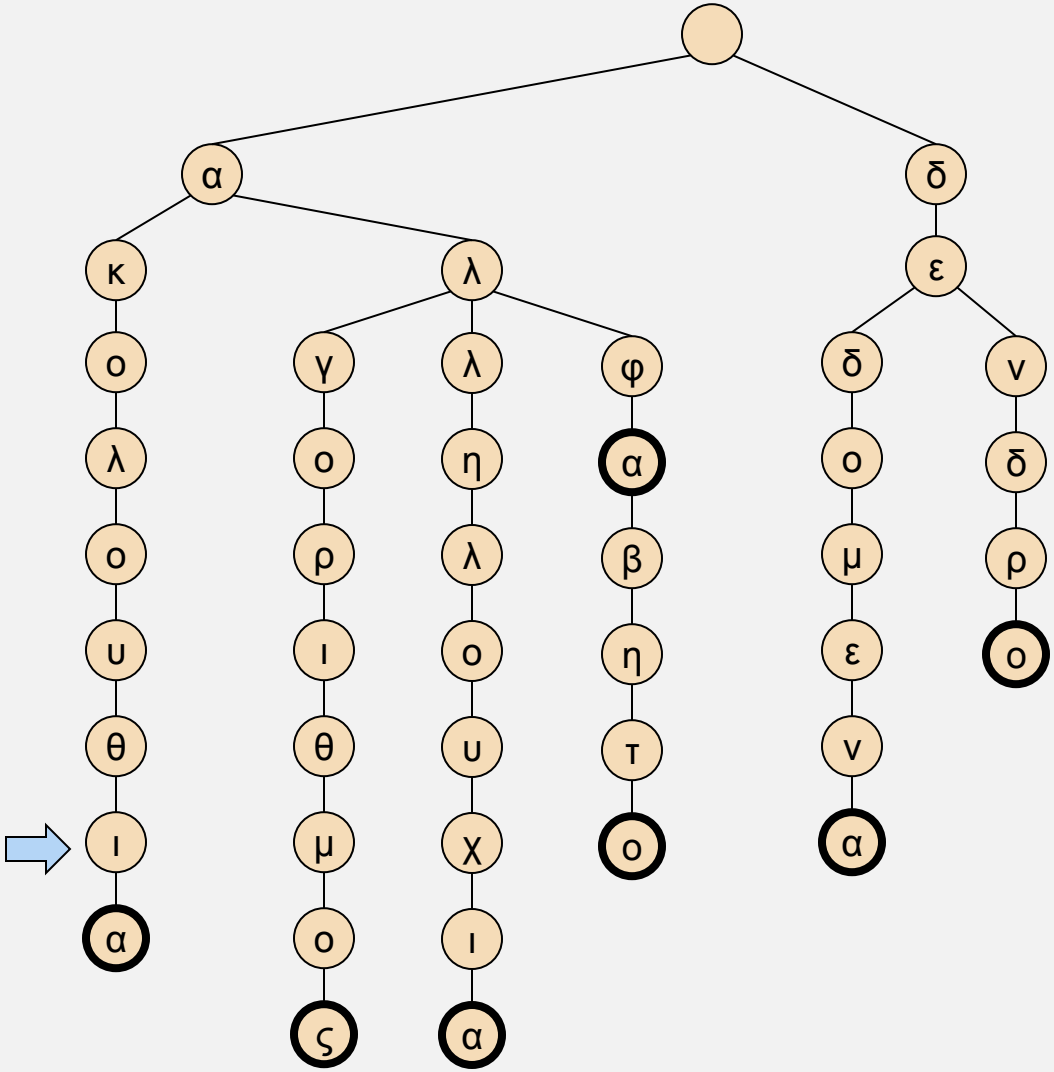
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “ακολουθι”
```

```
Queue q = ()
```



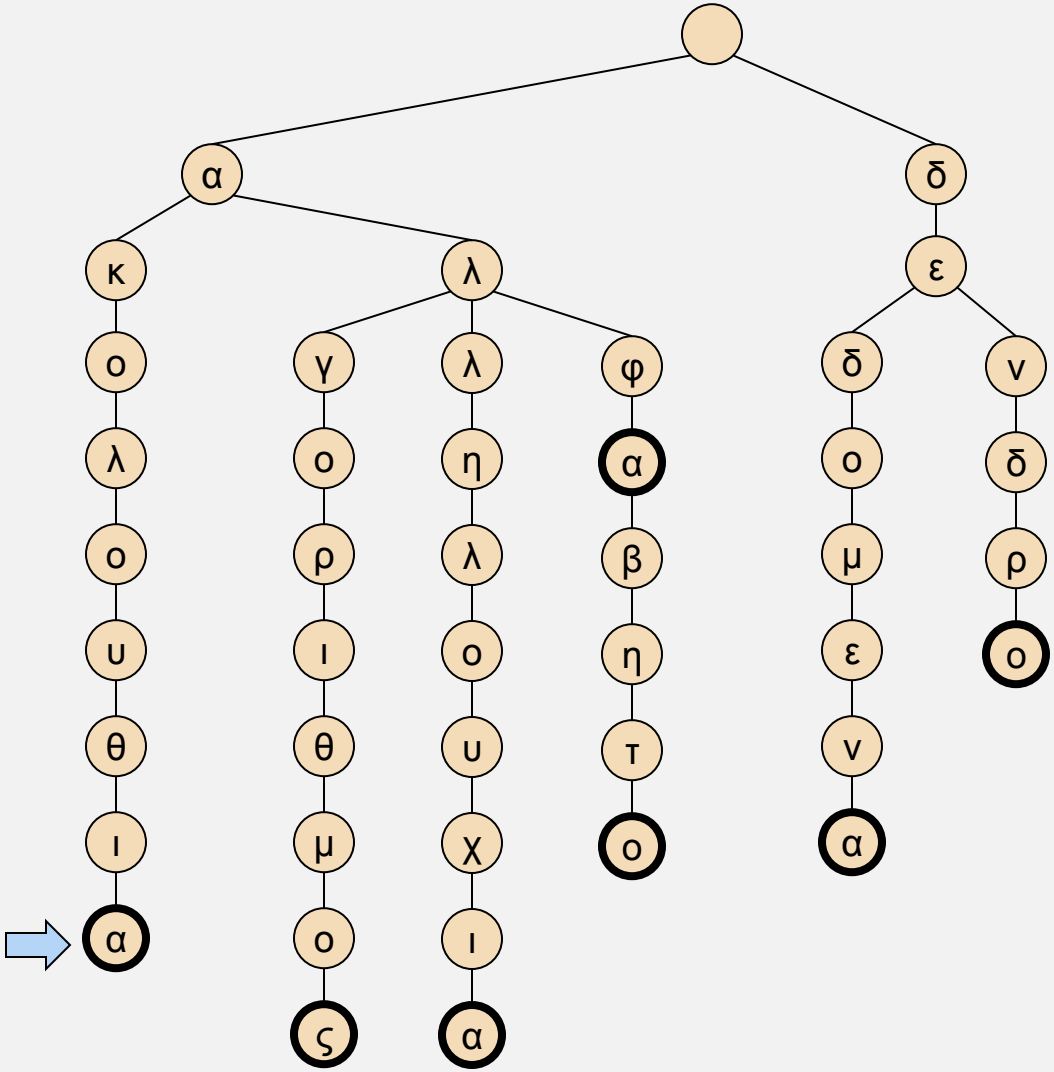
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “ακολουθια”
```

```
Queue q = (“ακολουθια”)
```



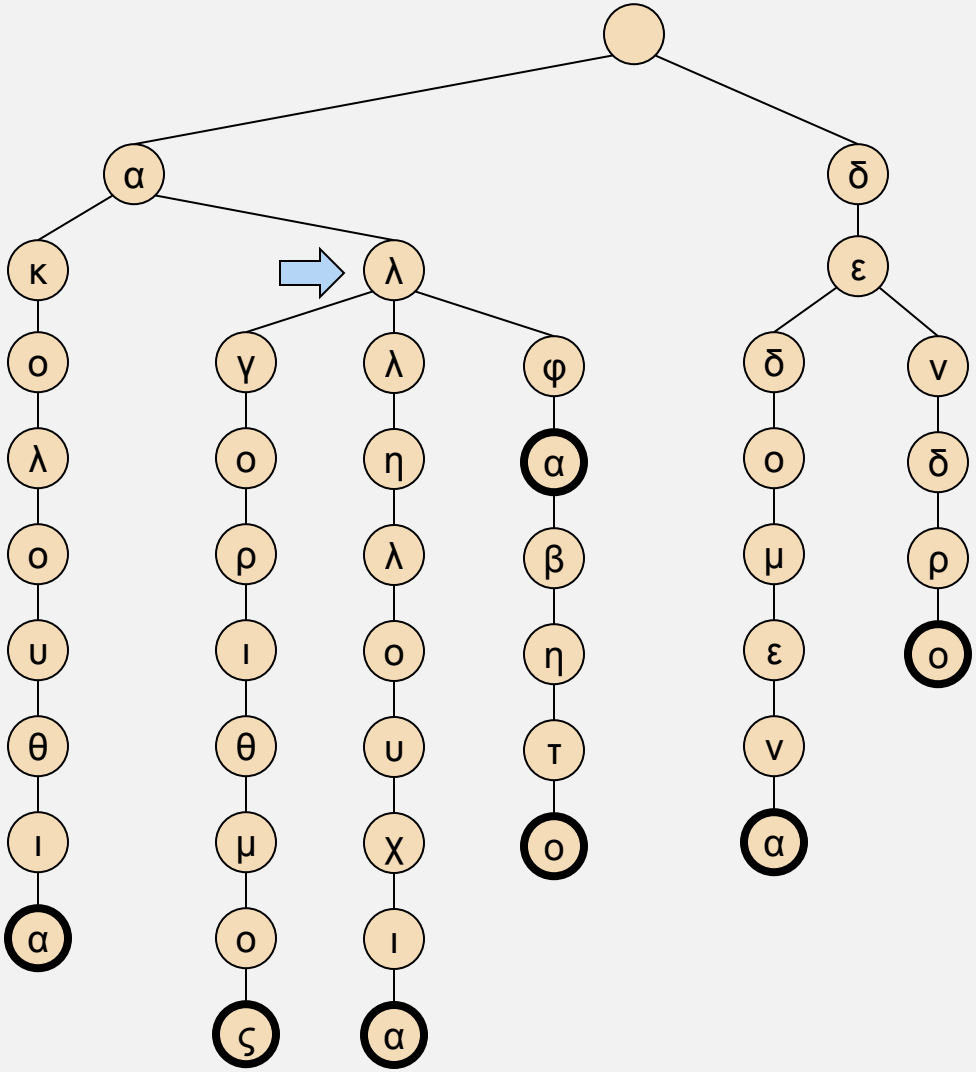
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “αλ”
```

```
Queue q = (“ακολουθια”)
```



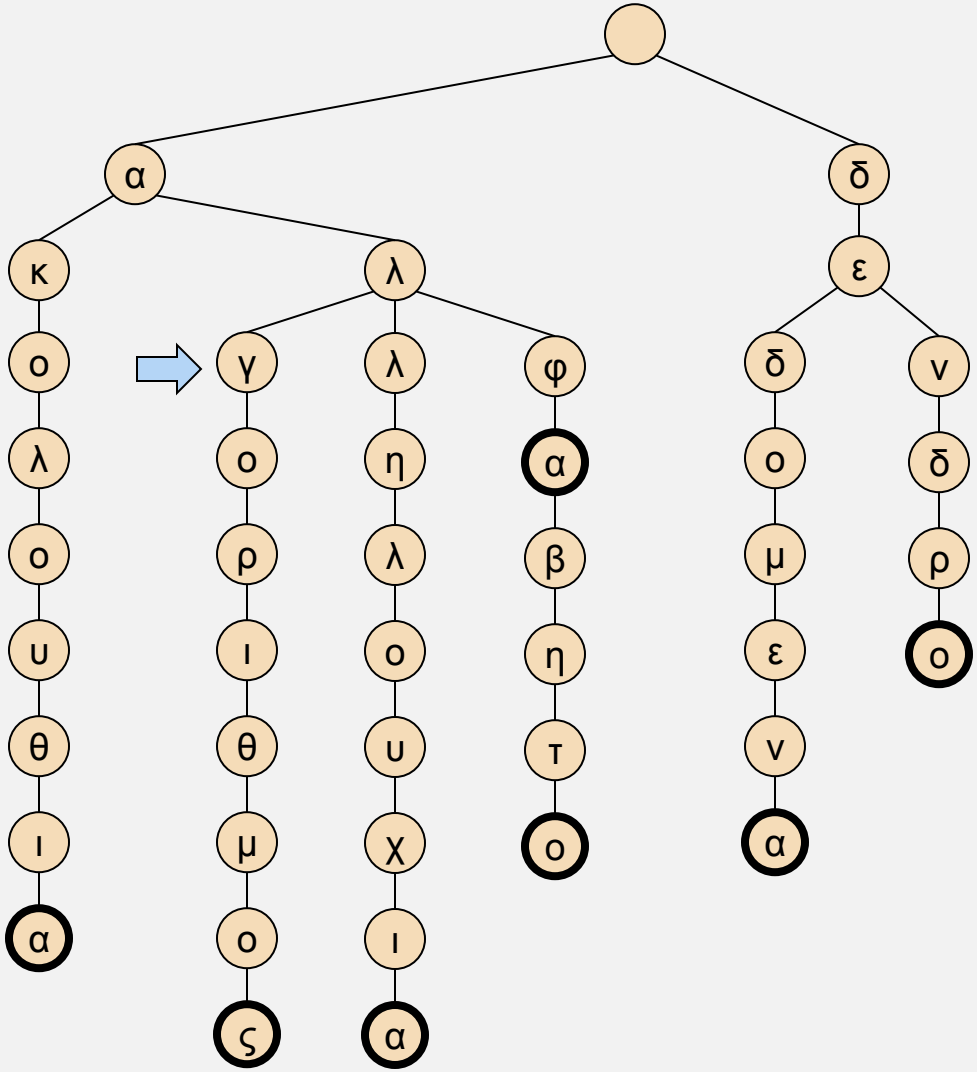
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “αλγ”
```

```
Queue q = (“ακολουθια”)
```



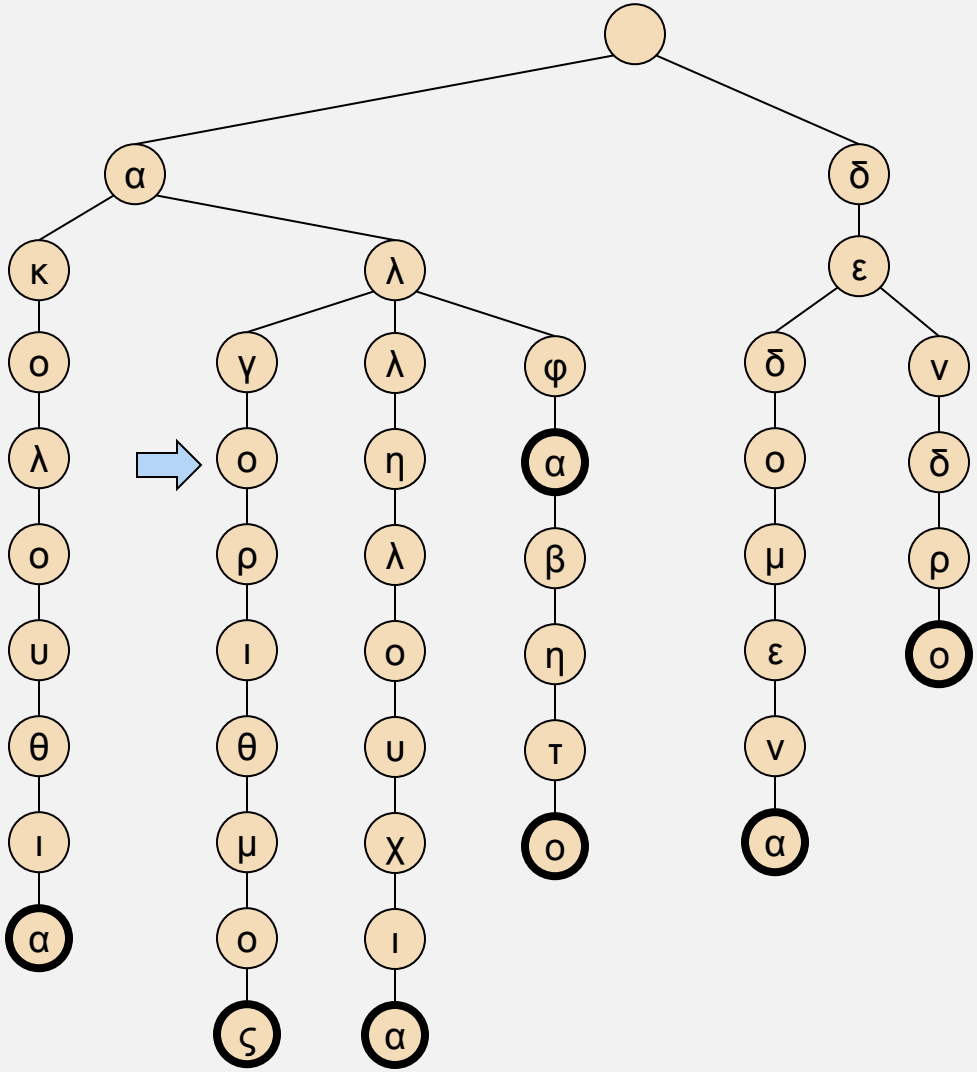
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = “αλγο”
```

```
Queue q = (“ακολουθια”)
```



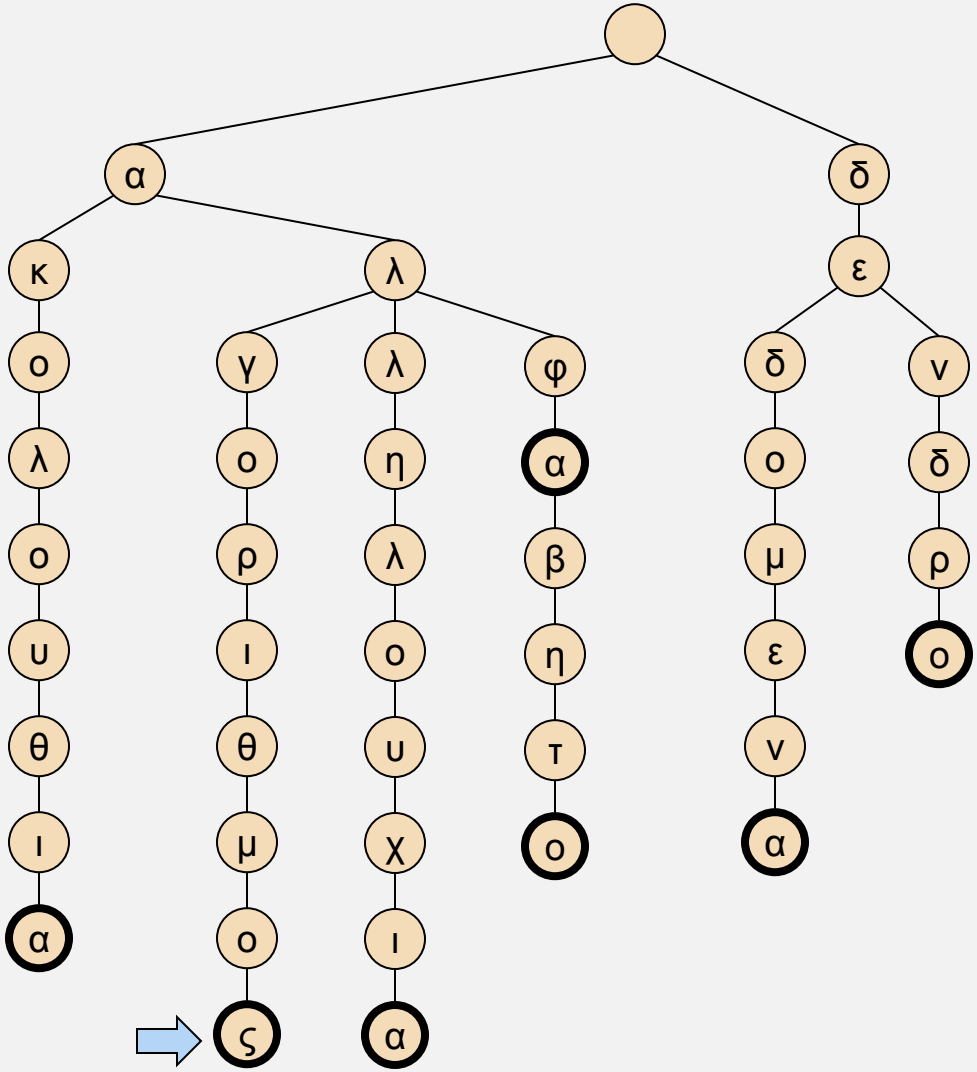
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = "αλγοριθμος"
```

```
Queue q = ("ακολουθια",
           "αλγοριθμος")
```



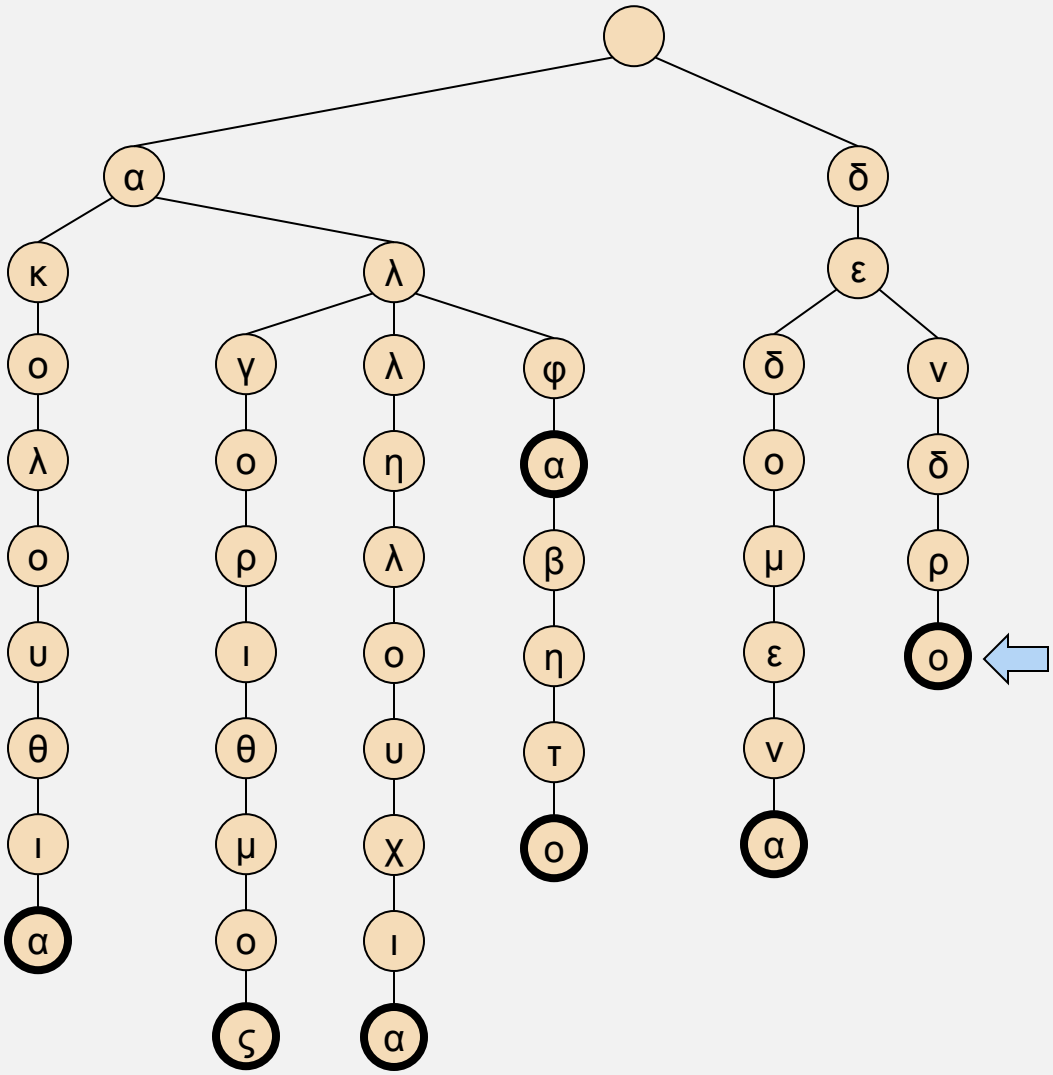
Tries για αποθήκευση αλφαριθμητικών

```
Iterable<String> keys()
```

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

```
pre = "δενδρο"
```

```
Queue q = ("ακολουθια",
           "αλγοριθμος",
           "αλληλουχια",
           "αλφα",
           "αλφαβητο",
           "δεδομενα",
           "δενδρο")
```

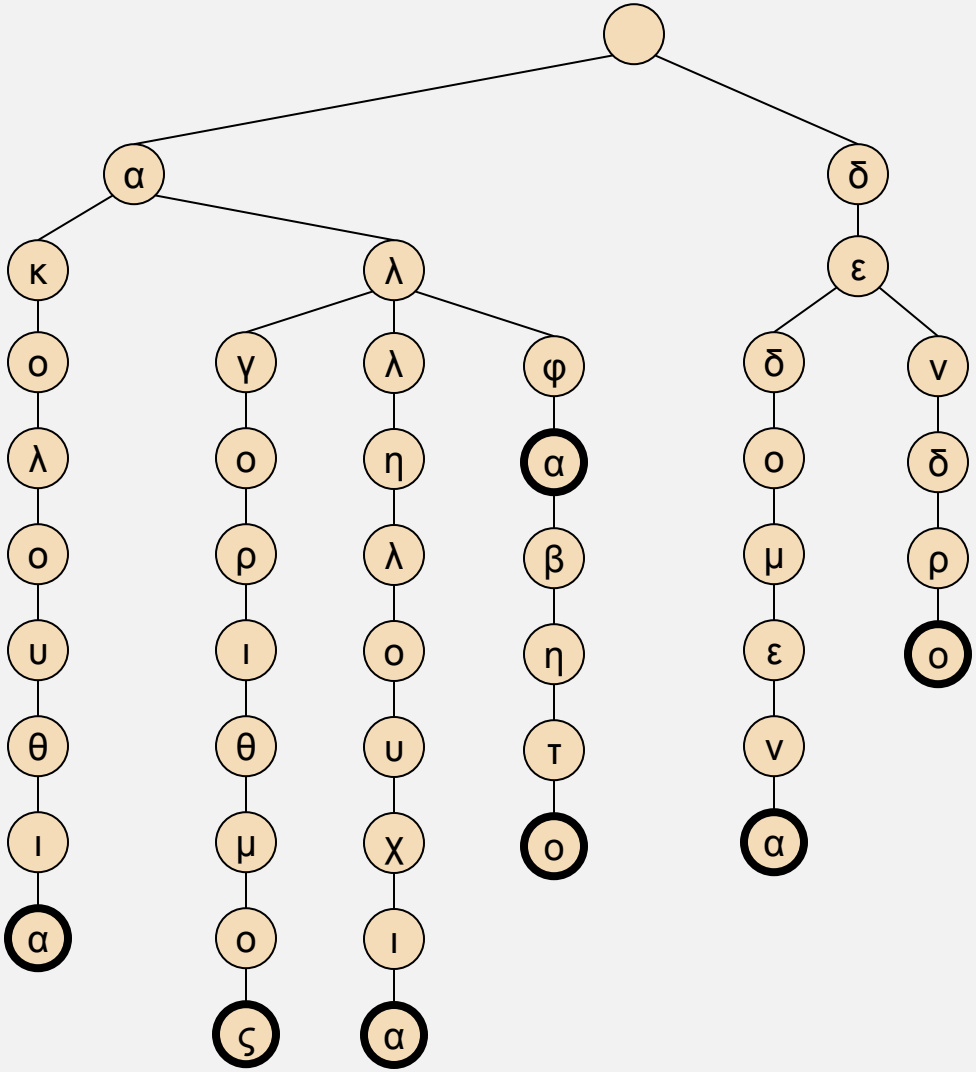


Tries για αποθήκευση αλφαριθμητικών

`Iterable<String> keys()`

Επισκεπτόμαστε τους κόμβους σε προδιάταξη και δημιουργούμε το αλφαριθμητικό που αντιστοιχεί στο μονοπάτι από τη ρίζα. Αν ο τρέχων κόμβος έχει επισήμανση τότε αποθηκεύουμε το αλφαριθμητικό σε μια ουρά.

Μπορούμε να υλοποιήσουμε με όμοιο τρόπο τις μεθόδους `keysWithPrefix()` και `keysThatMatch()`



Tries για αποθήκευση αλφαριθμητικών

```
public class StringTrie {
    private static int R = 256;  // πλήθος διαφορετικών χαρακτήρων
    private Node root;
    private static class Node {
         private boolean mark;  // bit επισήμανσης κόμβου (true αν είναι τέλος λέξης)
         private Node[] next = new Node [R];  // σύνδεσμοι σε R παιδιά
    }

    …

    public Iterable<String> keys()
    {
        Queue<String> q = new Queue<String>(); // αποθηκεύει τις λέξεις του trie
        collect(root, “”, q);
        return q;
    }

    private void collect(Node x, String pre, Queue<String> q)
    {   // αποθηκεύει τις λέξεις στο υποδένδρο του x
        if ( x==null ) return;
        if ( x.mark ) q.insert(pre);
        for (char c=0; c<R; c++)
            collect(x.next[c], pre+c, q);
    }
}
```

Tries για αποθήκευση αλφαριθμητικών

- Ένα trie κατασκευασμένο από n στοιχεία μέσου μήκους l από αλφάβητο R χαρακτήρων έχει πλήθος συνδέσμων μεταξύ $R \cdot n$ και $R \cdot n \cdot l$.

Απαιτεί υπερβολικά πολύ χώρο μνήμης για στοιχεία με μεγάλο μήκος με χαρακτήρες από μεγάλο αλφάβητο!

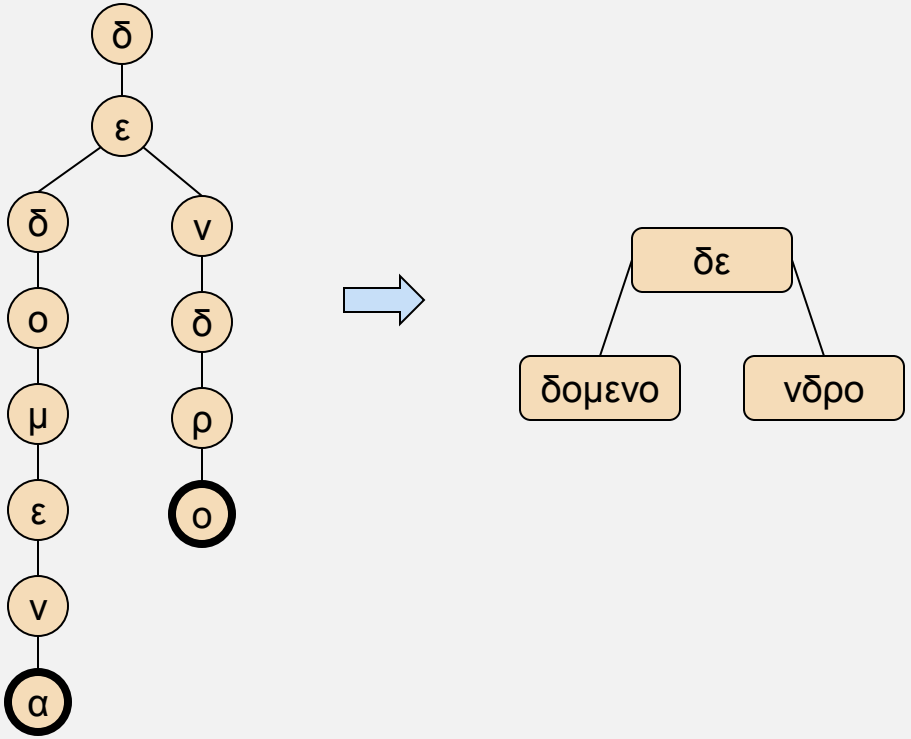
Tries για αποθήκευση αλφαριθμητικών

- Ένα trie κατασκευασμένο από n στοιχεία μέσου μήκους l από αλφάβητο R χαρακτήρων έχει πλήθος συνδέσμων μεταξύ $R \cdot n$ και $R \cdot n \cdot l$.

Απαιτεί υπερβολικά πολύ χώρο μνήμης για στοιχεία με μεγάλο μήκος με χαρακτήρες από μεγάλο αλφάβητο!

Μπορούμε, όμοια με τα δυαδικά trie, να αποθηκεύσουμε αλφαριθμητικά στους κόμβους .

Η λύση αυτή αντιμετωπίζει το πρόβλημα χώρου, αλλά ο χειρισμός του trie γίνεται πιο περίπλοκος.

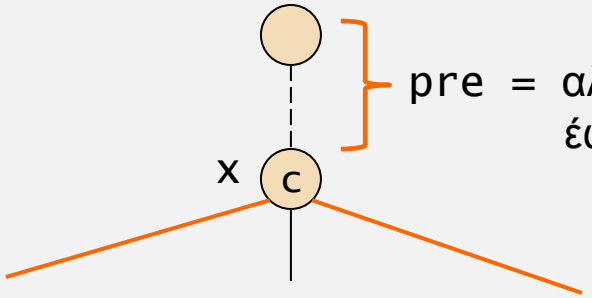


Tries για αποθήκευση αλφαριθμητικών

- Ένα trie κατασκευασμένο από n στοιχεία μέσου μήκους l από αλφάβητο R χαρακτήρων έχει πλήθος συνδέσμων μεταξύ $R \cdot n$ και $R \cdot n \cdot l$.

Απαιτεί υπερβολικά πολύ χώρο μνήμης για στοιχεία με μεγάλο μήκος με χαρακτήρες από μεγάλο αλφάβητο!

Εναλλακτική λύση : Τριαδικό trie – κάθε κόμβος έχει 3 συνδέσμους



σύνδεσμος προς αλφαριθμητικά με πρόθεμα `pre` και επόμενο χαρακτήρα `< c`

σύνδεσμος προς αλφαριθμητικά με πρόθεμα `pre+c`

σύνδεσμος προς αλφαριθμητικά με πρόθεμα `pre` και επόμενο χαρακτήρα `> c`

Τριαδικό Trie

Παράδειγμα:

Τριαδικό trie για τις λέξεις

ακολουθία
αλληλουχία
αλγόριθμος
αλφάβητο
άλφα
δεδομένα
δένδρο

Οι κόμβοι που αντιστοιχούν στο τέλος μιας λέξης είναι επισημασμένοι