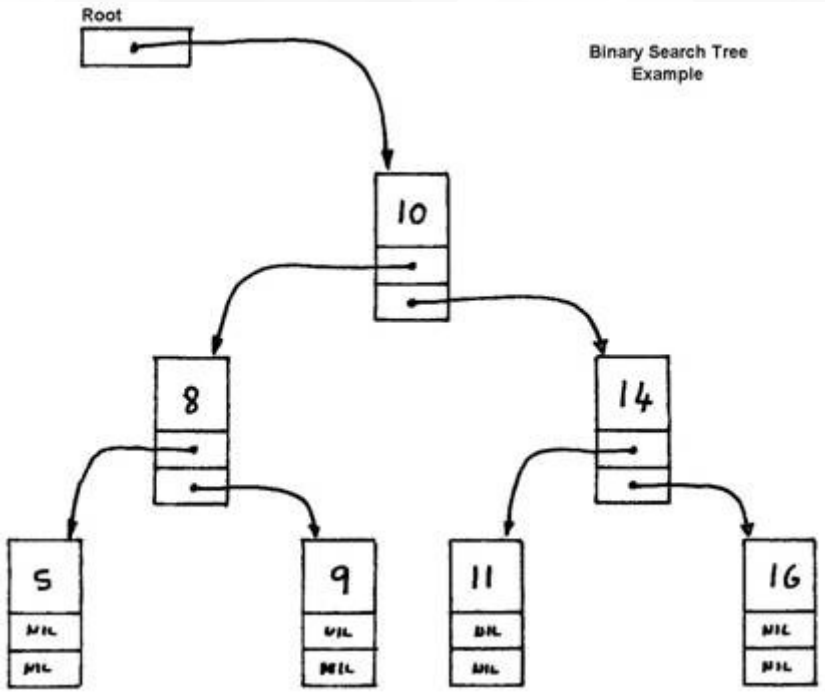
MC3305
Algoritmos e Estruturas de Dados II

Aula 15 –
Árvores Digitais / Trie

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

2Q-2015

ABBs



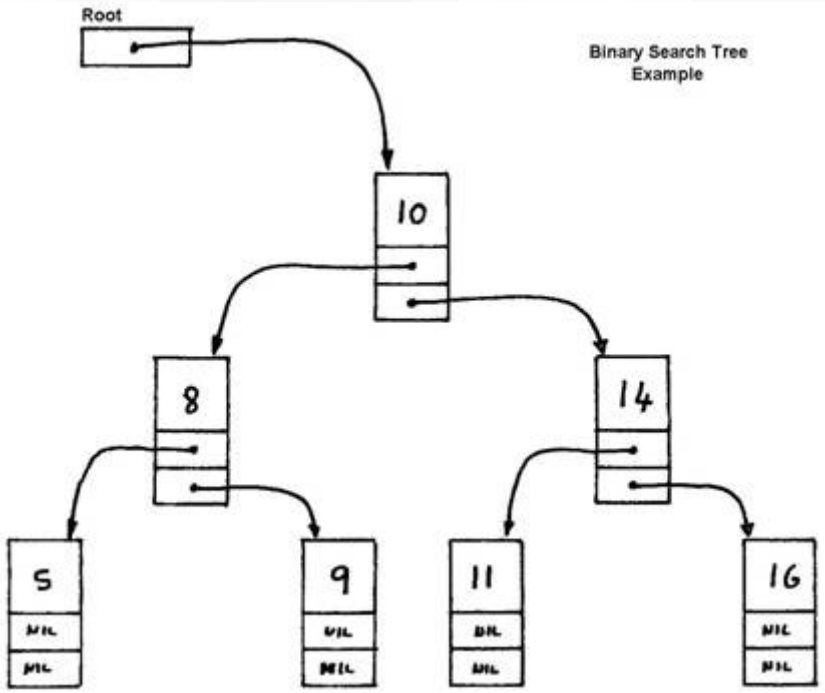
Problema de busca geral

- Conjunto de chaves (S).
- Elemento x a buscar em S.

Até agora vimos que:

- As chaves são elementos indivisíveis.
- As chaves tem mesmo tamanho.

ABBs



Problema de busca geral

- Conjunto de chaves (S).
- Elemento x a buscar em S.

Até agora vimos que:

- As chaves são elementos indivisíveis.
- As chaves tem mesmo tamanho.

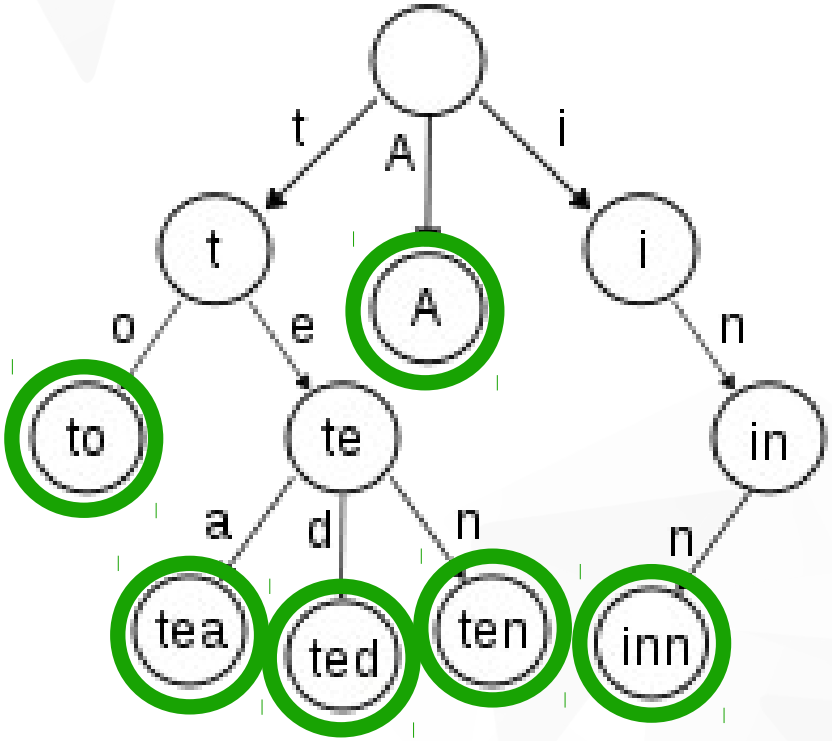
E se a busca consistir em frases / texto literário?

Árvores digitais (Árvore de prefixos)

Todos os descendentes de um nó tem um prefixo em comum

Palavras/Chaves:

- A
- to
- tea
- ted
- ten
- inn

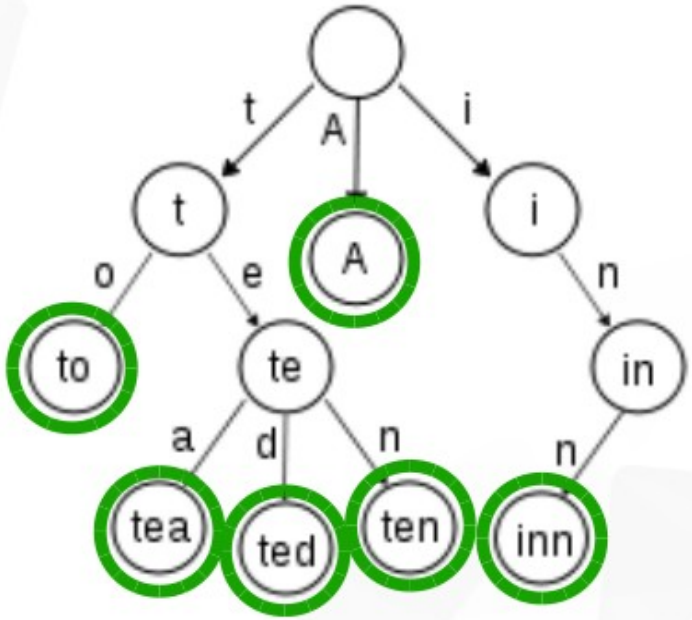


Palavras com tamanho variável e ilimitado

Árvores digitais (Árvore de prefixos)

- Uma árvore TRIE (ATRIE) é uma **estrutura de dados do tipo árvore ordenada que permite a recuperação de informação**.

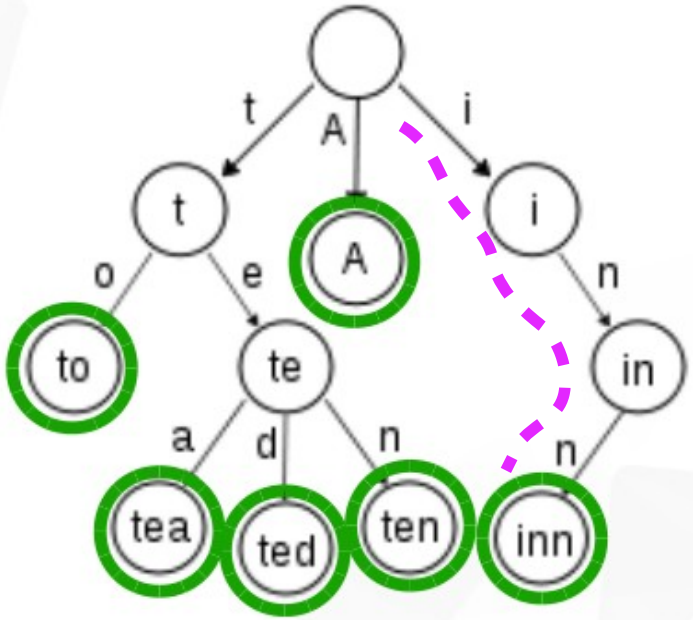
- A ATRIE é **utilizada para armazenar um array associativo** em que as chaves são normalmente cadeias de caracteres



Árvores digitais (Árvore de prefixos)

- Uma árvore TRIE (ATRIE) é uma **estrutura de dados do tipo árvore ordenada que permite a recuperação de informação**.

- A ATRIE é **utilizada para armazenar um array associativo** em que as chaves são normalmente cadeias de caracteres



A chave é uma sequencia de símbolos pertencentes a um alfabeto.

As **chaves são armazenadas nas folhas da árvore**, os nós internos são parte do caminho que direciona a busca.

Compara digitos da chave individualmente

TRIE originado de 'Information reTRIEval'

TRIE = digital tree = radix tree = prefix tree

Trie Memory*

Edward Fredkin, *Bolt Beranek and Newman, Inc., Cambridge, Mass.*

Introduction

Trie memory is a way of storing and retrieving information.[1] It is applicable to information that consists of function-argument (or item-term) pairs—information conventionally stored in unordered lists, ordered lists, or pigeonholes.

The main advantages of trie memory over the other memory plans just mentioned are shorter access time, greater ease of addition or up-dating, greater convenience in handling arguments of diverse lengths, and the ability to take advantage of redundancies in the information stored. The main disadvantage is relative inefficiency in using storage space, but this inefficiency is not great when the store is large.

In this paper several paradigms of trie memory are described and compared with other memory paradigms, their advantages and disadvantages are examined in detail, and applications are discussed.

is simply the binary variable of which the admissible values are *member* and *nonmember*.

At the outset, before a storage is begun, the trie is merely a collection of *registers*. Except for two special registers, which we may call α and δ, every register has a *cell* for each member (type) of the ensemble of alphabetic characters. If we let that ensemble include a "space" to indicate the end of a word (argument), each register must have 27 cells.

Each cell has space for the address of any register in the memory. Cells in the trie that are not yet being used to represent stored information always contain the address of the special α register. A cell thus represents stored information if it contains the address of some register other than α. The information it represents is its own name, "A" for the A cell, "B" for the B cell, etc., and the address of the next register in the sequence.

Storage of words of alphabetic characters is illustrated in Fig. 1. For the sake of simplicity, the ensemble of

Published in:

· Magazine
Communications of the ACM CACM Homepage archive
Volume 3 Issue 9, September 1960
Pages 490-499
ACM New York, NY, USA
table of contents doi>10.1145/367390.367400

Edward Fredkin

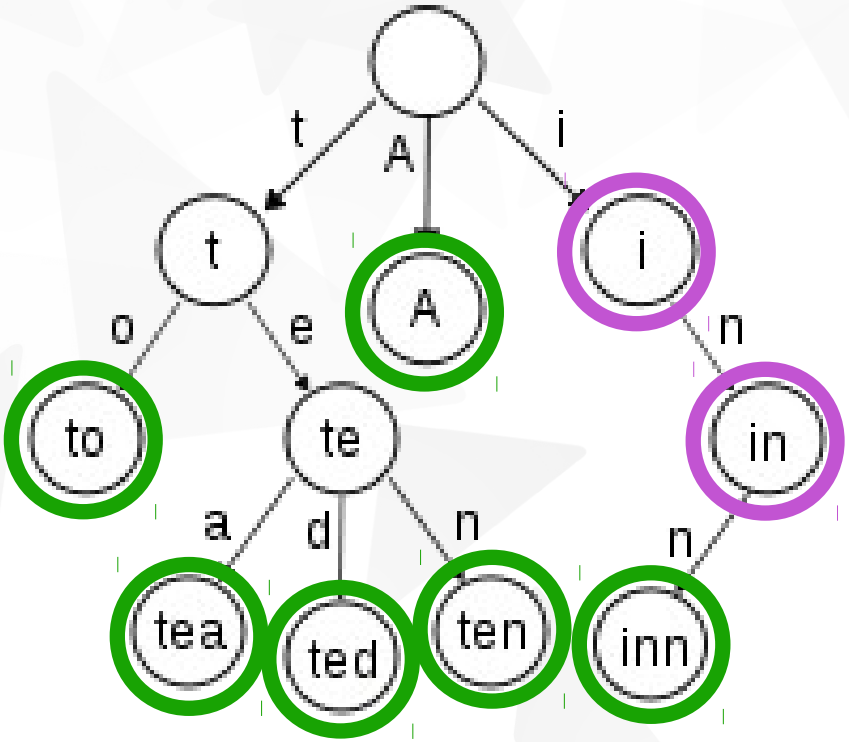
Born	October 2, 1934 (age 80)[citation needed] Los Angeles
Residence	Brookline, MA
Citizenship	USA
Nationality	USA
Fields	Computer Science, Physics. Business,
Institutions	Massachusetts Institute of Technology Carnegie Mellon University Capital Technologies, Inc.
Alma mater	(Caltech)
Known for	Trie data structure, Fredkin gate
Notable awards	Dickson Prize in Science 1984

Shannon + McCarthy + Fredkin + Waizenbaum

Claude Shannon, John McCarthy, Ed Fredkin and Joseph Weizenbaum (1966) [6]

[6] **Weizenbaum. Rebel at Work**. A documentary by Peter Haas and Silvia Holzinger
http://www.ilmarefilm.org/W_E_4_70.htm

Árvores TRIE

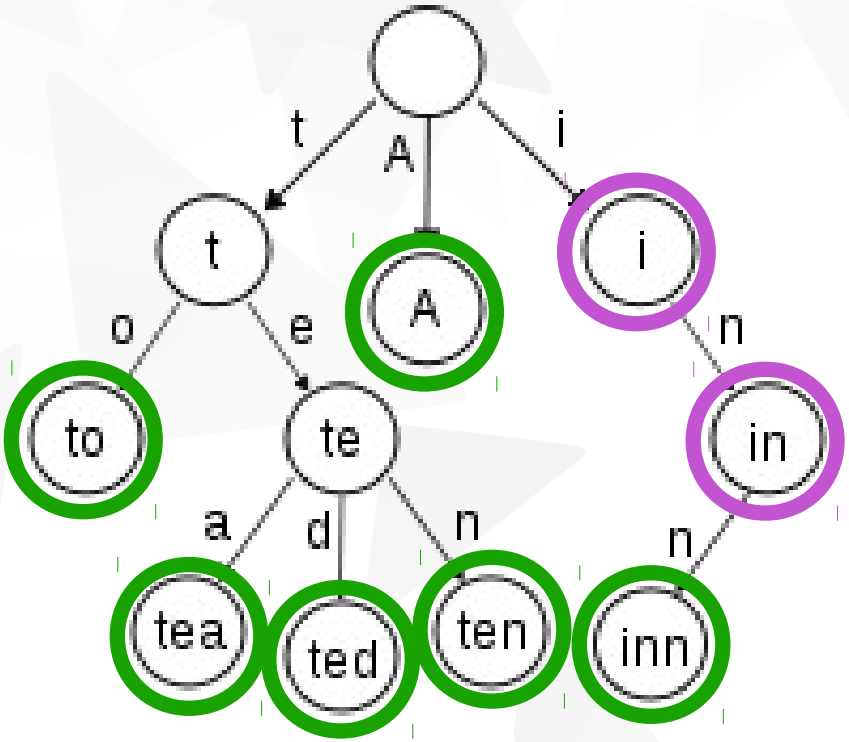


As **chaves são armazenadas nas folhas da árvore**, os nós internos são parte do caminho que direciona a busca (os nós internos também podem ser chaves).

Palavras/Chaves:

- **A**
- **to**
- **tea**
- **ted**
- **ten**
- **Inn**
- **i**
- **inn**

Árvores TRIE

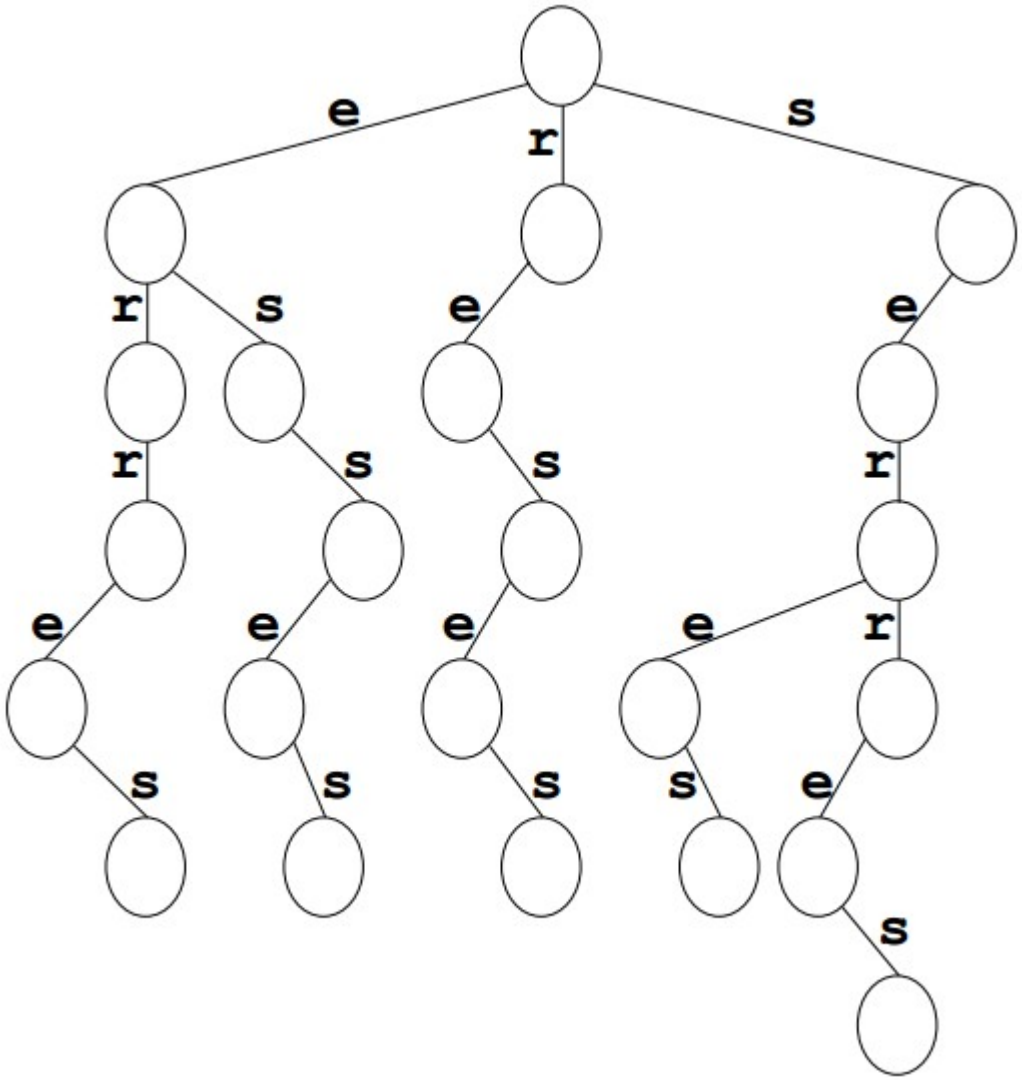


- A busca se inicia na raiz.
- As busca continua com a subárvore associado ao símbolo / caratere procurado até chegar a uma folha (ou nó interno)

Essa estrutura permite fazer buscas eficiente de cadeias que compartilham prefixo.

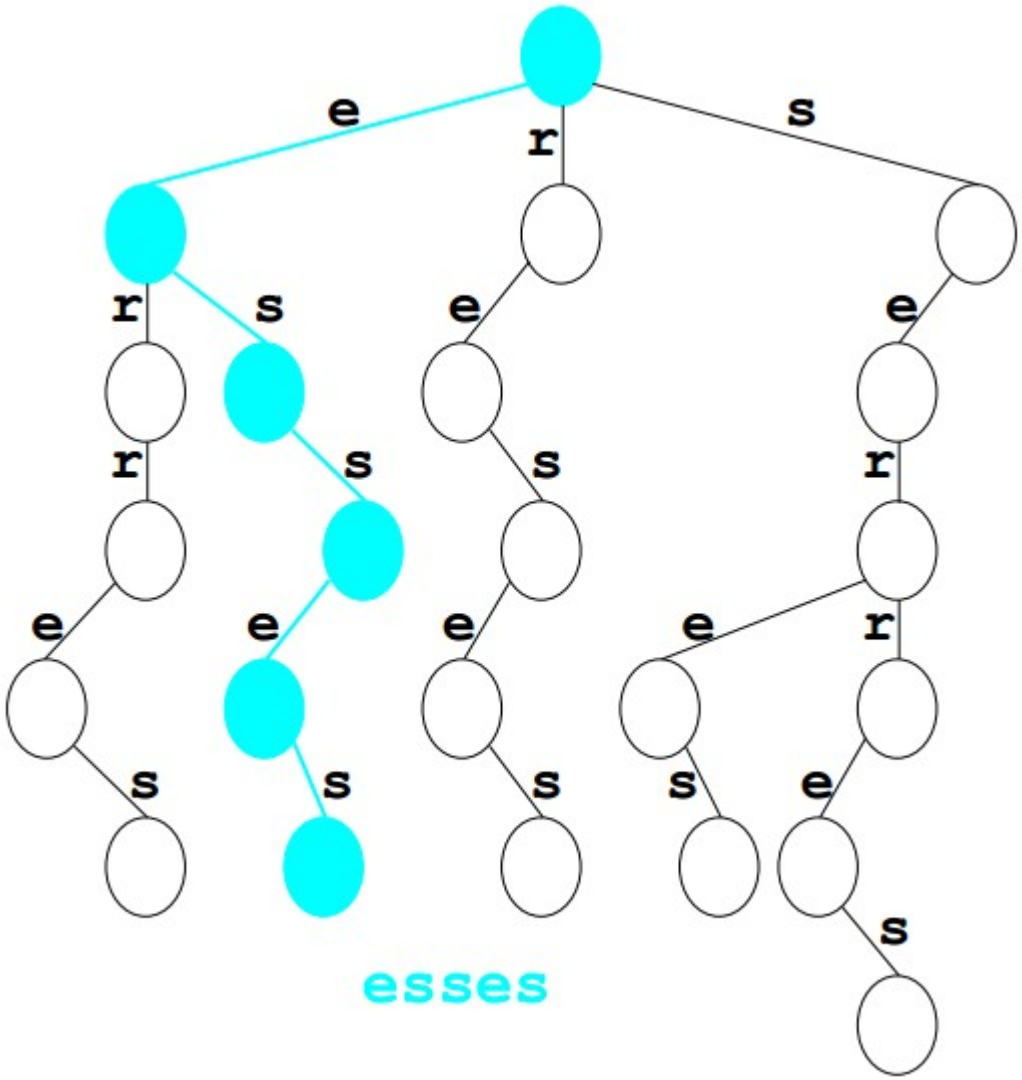
Exemplo de árvore TRIE

Alfabeto:
$\{e, r, s\}$



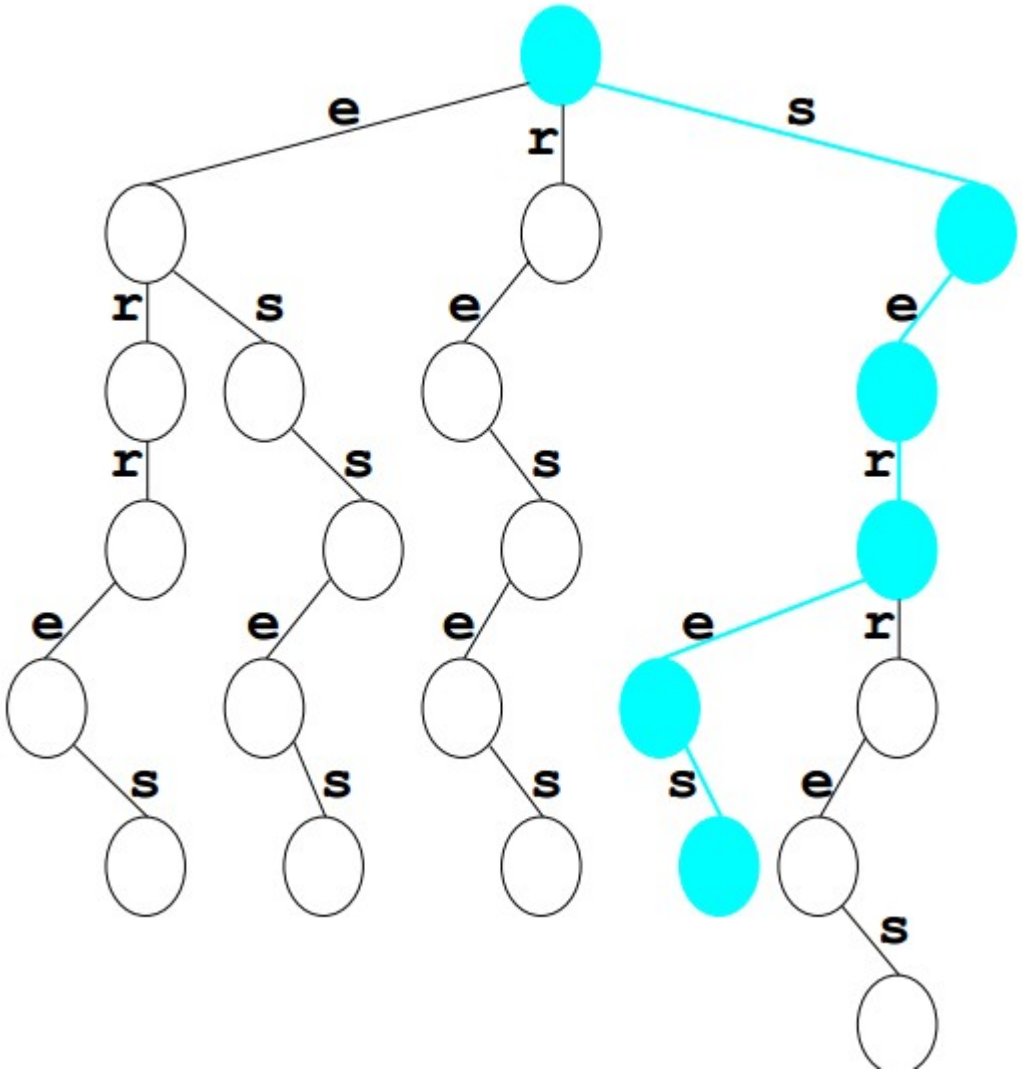
Exemplo de árvore TRIE

Alfabeto:
$\{e, r, s\}$



Exemplo de árvore TRIE

Alfabeto:
$\{e, r, s\}$



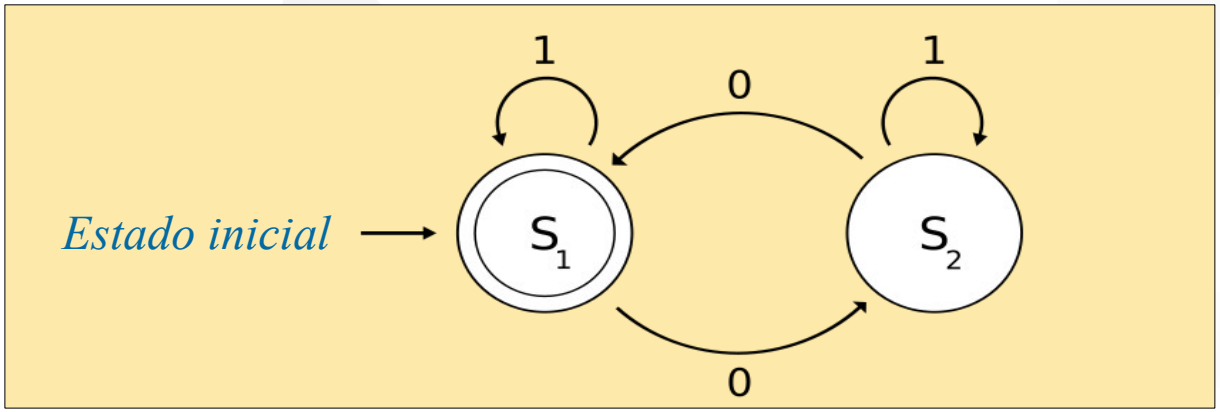
erre
erres
es
esse
esses
se
ser
serre
re
res
rese
reses
serres
seres

Árvore TRIE

- Uma árvore TRIE é um caso especial de um **autômato finito determinista (máquina de estados finito)** que sirve para armazenar/representar um conjunto de cadeias.

  A máquina está em apenas um estado por vez (este estado é chamado *estado atual*).

  O *estado de aceitação* é aquele em que a máquina relata que a sequência de entrada, como processada até agora, é membro do conjunto de cadeias aceitas.

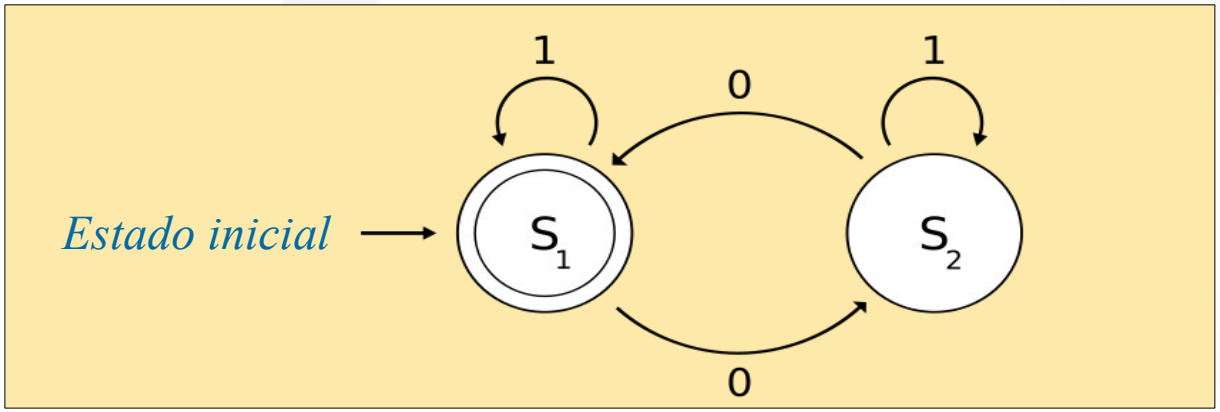


Árvore TRIE

- Uma árvore TRIE é um caso especial de um **autômato finito determinista (máquina de estados finito)** que serve para armazenar/representar um conjunto de cadeias.

A máquina está em apenas um estado por vez (este estado é chamado *estado atual*).

O *estado de aceitação* é aquele em que a máquina relata que a sequência de entrada, como processada até agora, é membro do conjunto de cadeias aceitas.



Este exemplo mostra um autômato que determina se um número binário tem **um número par ou ímpar de 0's**

Árvore TRIE

Definição formal da ATRIE para armazenar um conjunto de cadeia E:

$$(S, \Sigma, T, s, A)$$

- S é o conjunto de estados, cada um dos quais representa um prefixo de E.
- Σ é o alfabeto sobre o qual estão definidas as cadeias.

Árvore TRIE

Definição formal da ATRIE para armazenar um conjunto de cadeia E:

$$(S, \Sigma, T, s, A)$$

- S é o conjunto de estados, cada um dos quais representa um prefixo de E.
- Σ é o alfabeto sobre o qual estão definidas as cadeias.
- T é a função de transição de estados.

  $T(x, \sigma) = x\sigma$, se $x, x\sigma \in S$
- s corresponde ao estado inicial (igua à cadei vazia).
- A é o conjunto de estados de aceitação

  $A \subseteq S$, A = E

Exemplos de alfabetos e chaves

Alfabetos

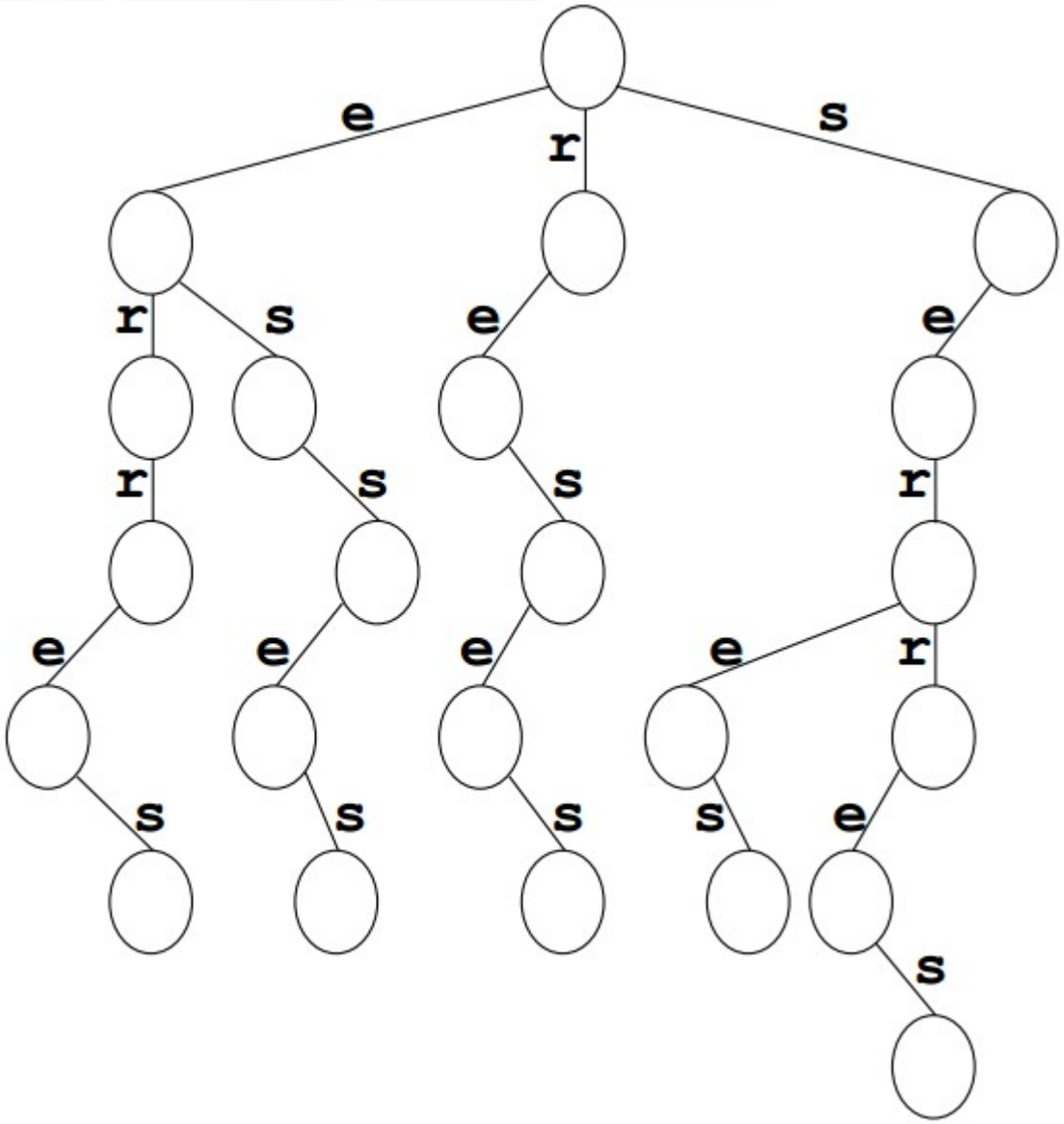
{0,1}, {A, B, C, D, E,...Z}, {0,1,2,3,4,5,...,9}

Chaves

ABABBBABABA 19034717 Maria
01010101000000000101000000001010

A chave é determinada pela posição na árvore.

Árvore TRIE (árvore m-ária)



Árvore ternária;

Alfabeto:

$\{e, r, s\}$ $e < r < s$

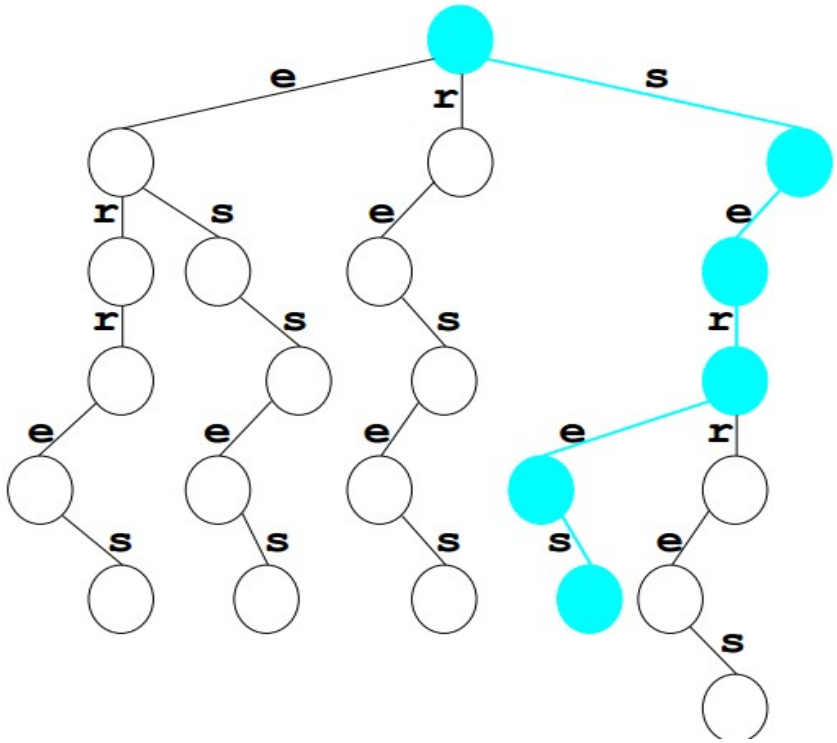
$m = 3$

$S = \{erre, erres, es,$
$esse, esses, se,$
$ser, serre, re,$
$res, rese, reses,$
$serres, seres\}$

Busca digital

O método de busca digital é análogo à busca manual em dicionários:

Com a primeira letra da palavra são determinadas todas as páginas que contêm as palavras iniciadas por aquela letra e assim por diante.

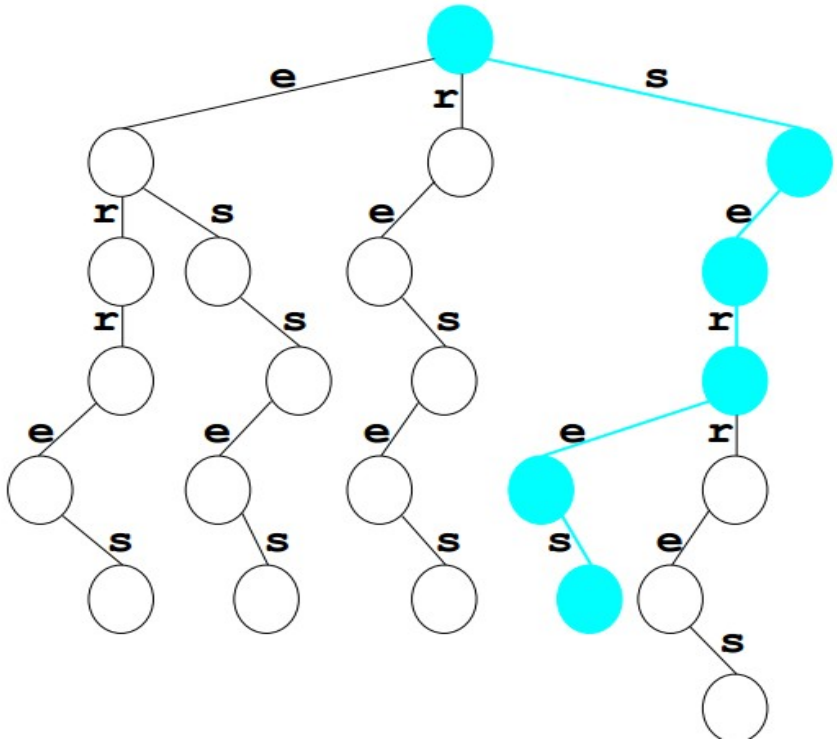


*A busca de uma chave de tamanho m, no pior caso terá um custo de **O(m)***

Busca digital

O método de busca digital é análogo à busca manual em dicionários:

Com a primeira letra da palavra são determinadas todas as páginas que contêm as palavras iniciadas por aquela letra e assim por diante.



A busca de uma chave de tamanho m, no pior caso terá um custo de ***O(m)***

Independe do número total de chaves.

Depende do tamanho da chave procurada.

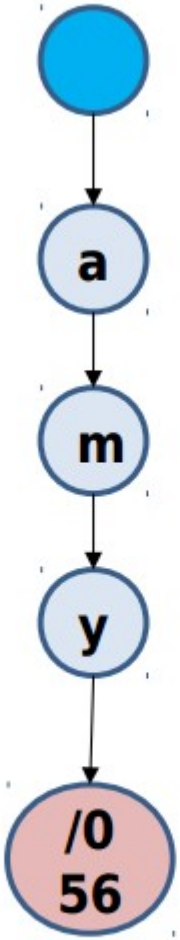
Montando uma árvore TRIE

Considere as seguintes chaves:

- amy 56
- ann 15
- emma 30
- rob 27
- roger 52

Montando uma árvore TRIE

- amy 56



<- Nível 0 (RAIZ)

<- Nível 1

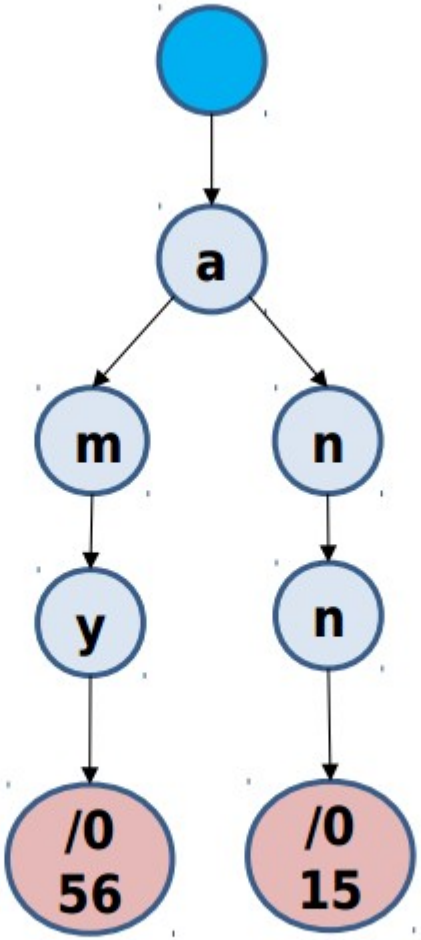
<- Nível 2

<- Nível 4

<- Nível 5

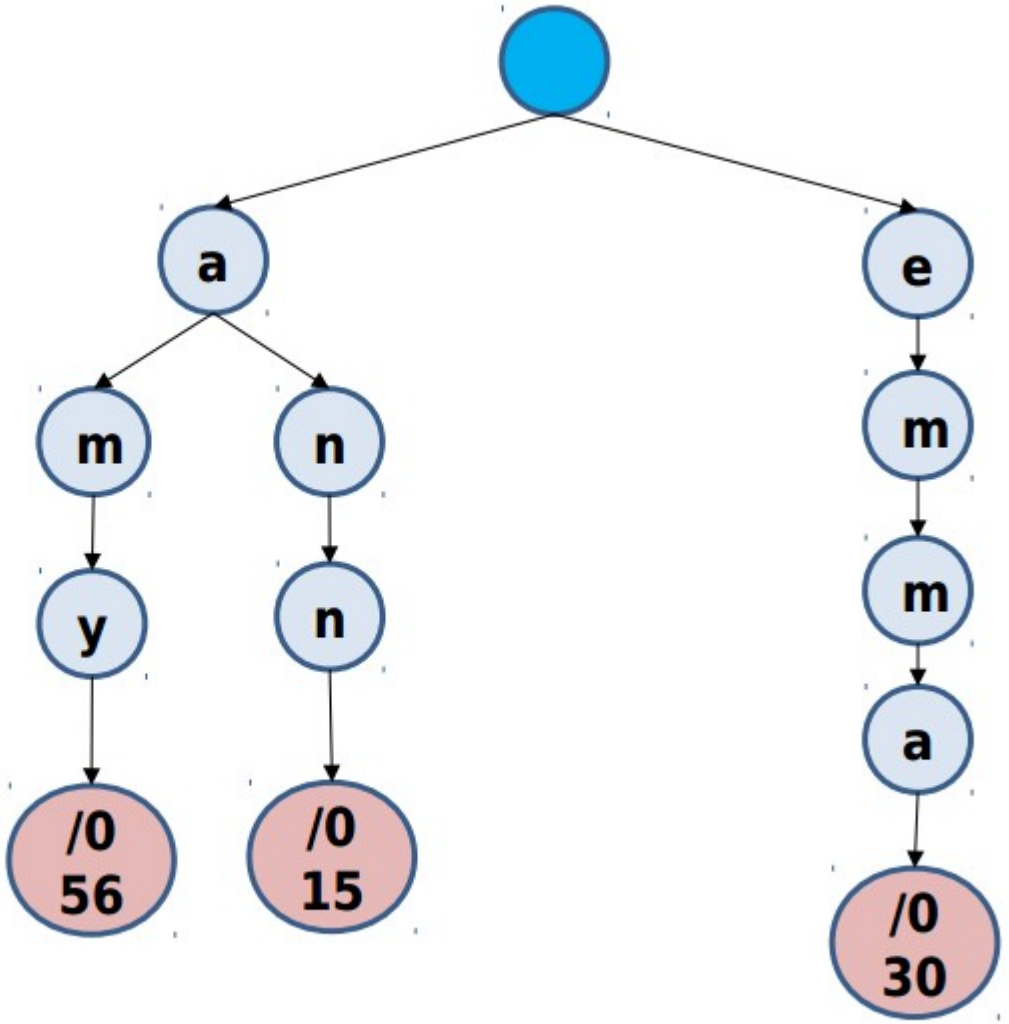
Montando uma árvore TRIE

- ann 15



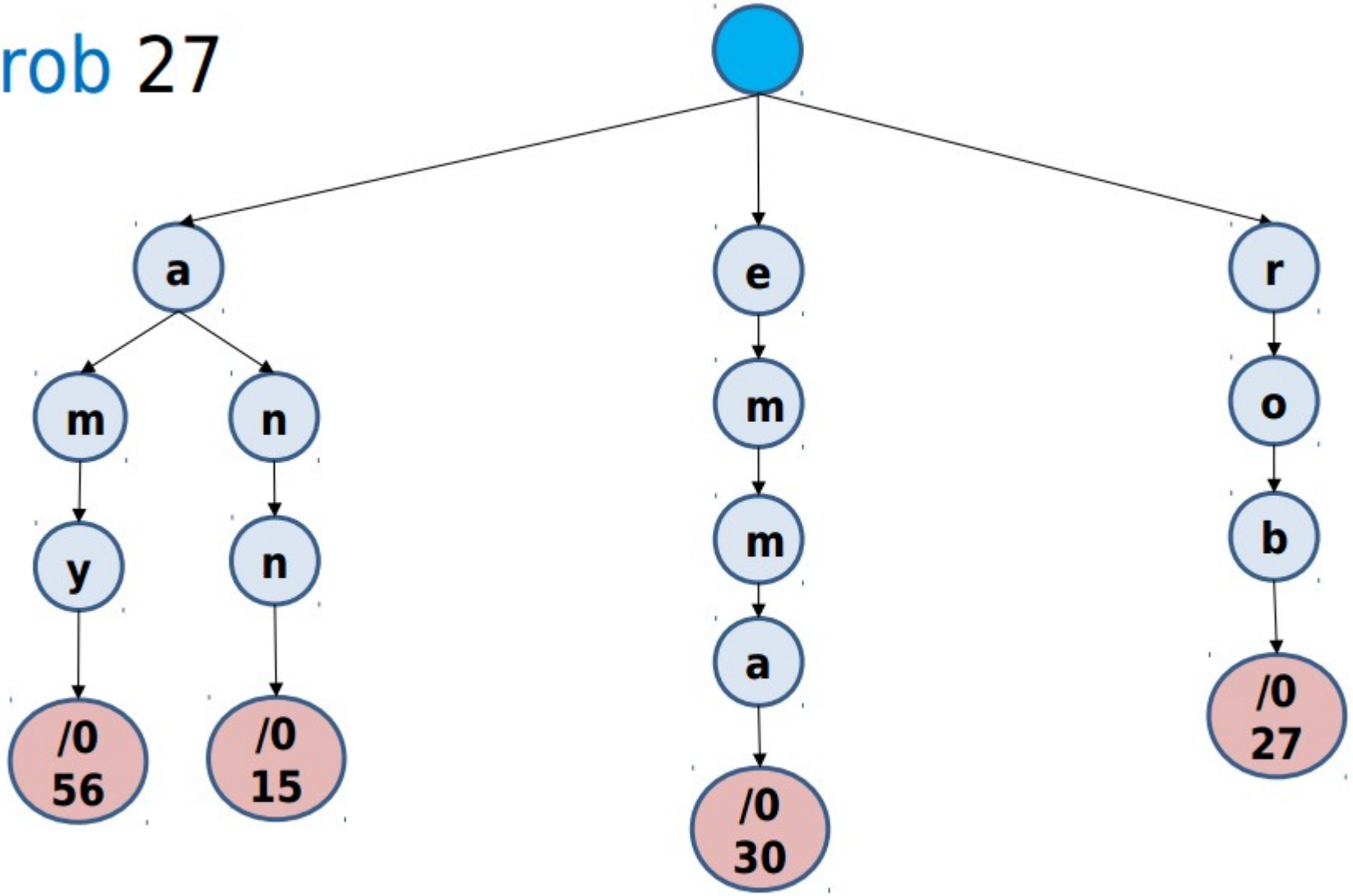
Montando uma árvore TRIE

- emma 30



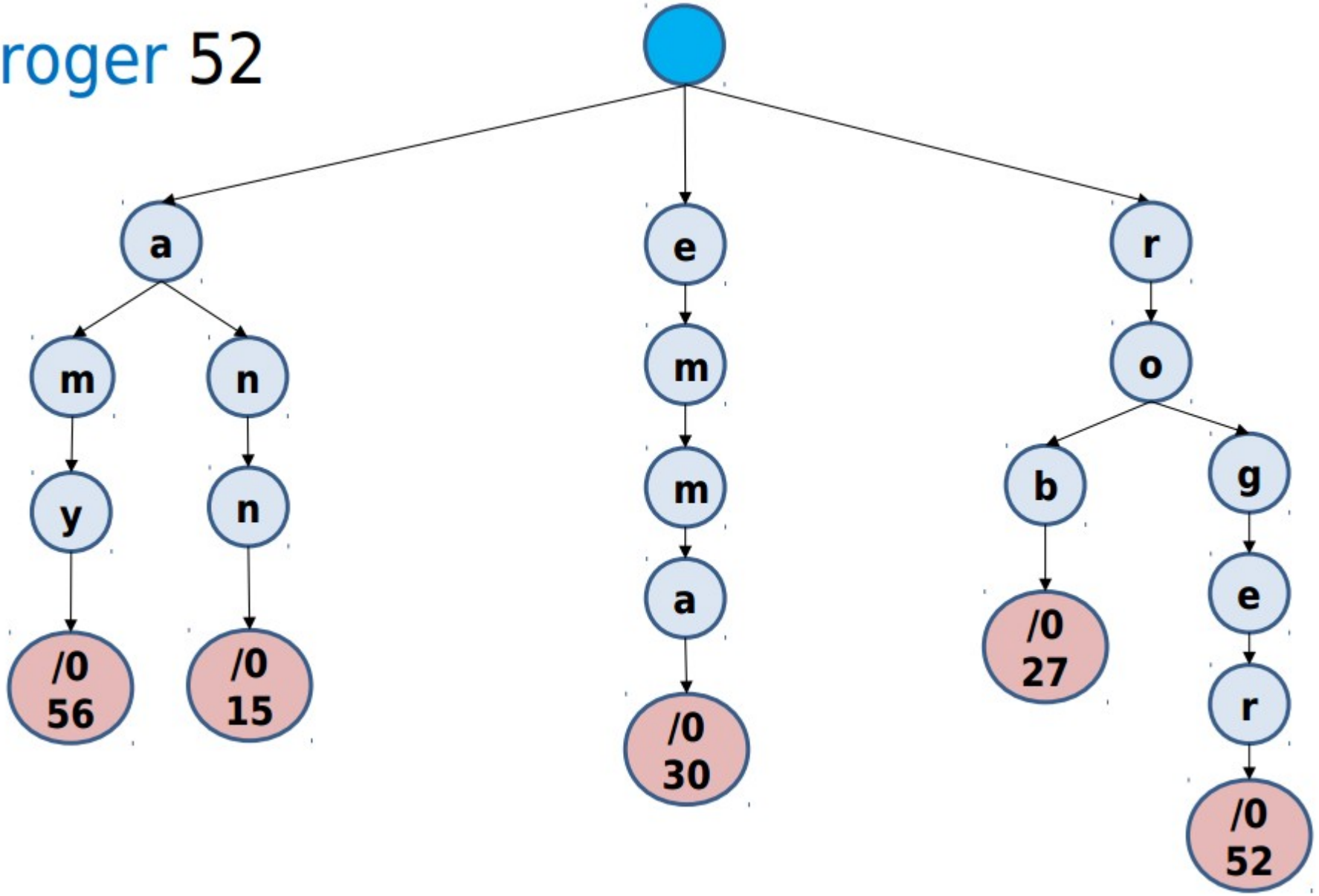
Montando uma árvore TRIE

- rob 27



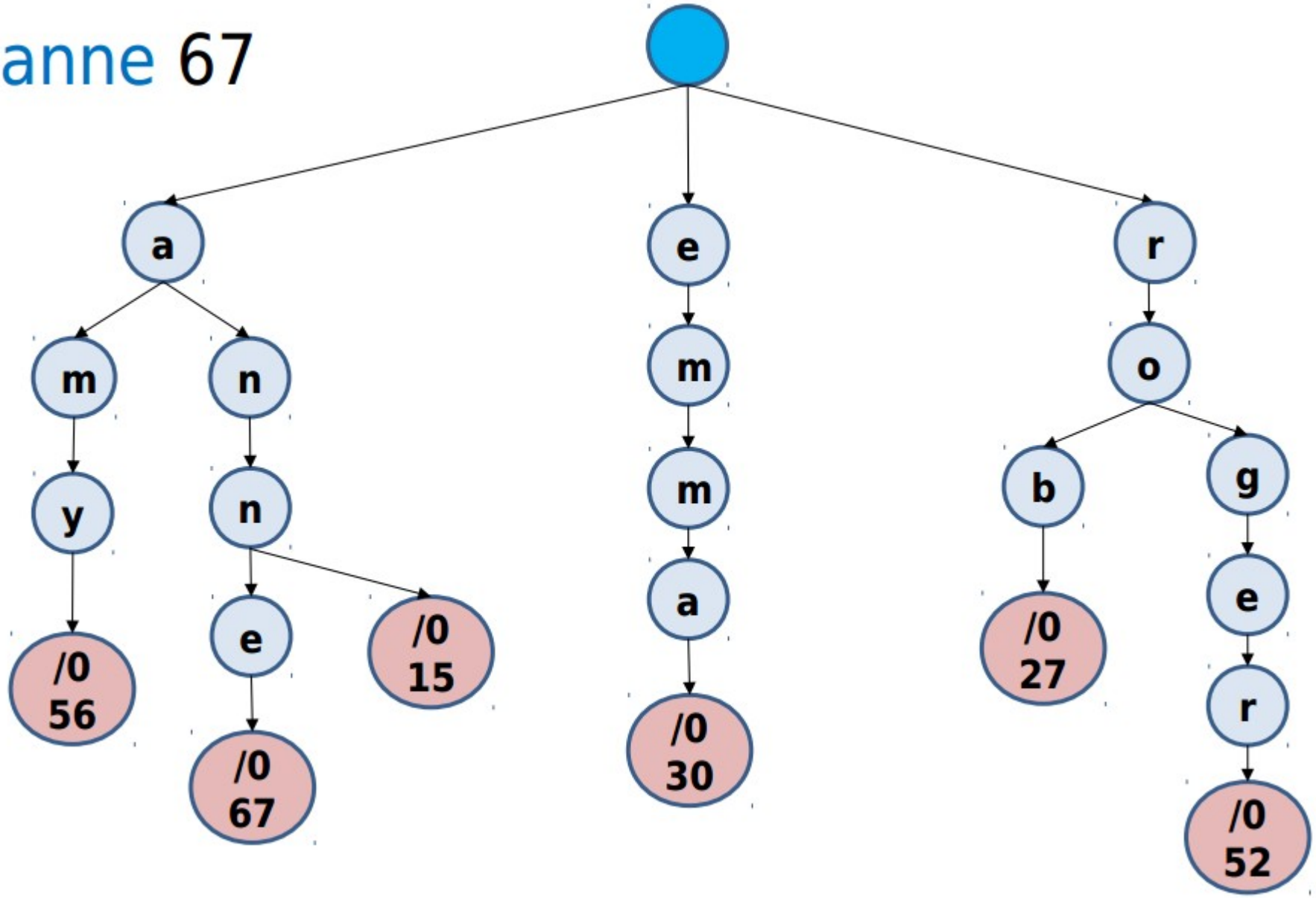
Montando uma árvore TRIE

- roger 52



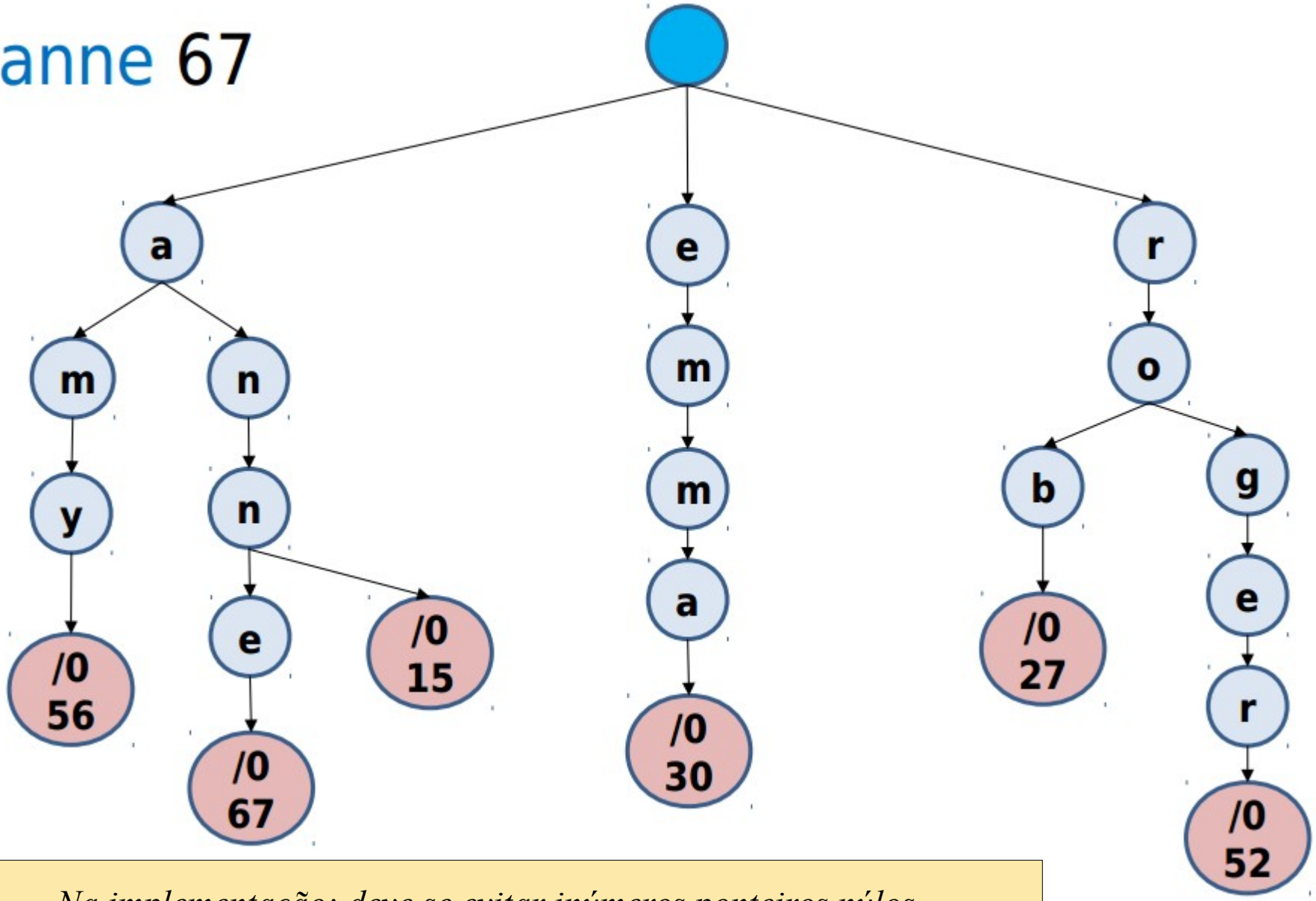
Montando uma árvore TRIE

- anne 67



Montando uma árvore TRIE

- anne 67



Na implementação: deve se evitar inúmeros ponteiros núlos.

Implementação

Implementando uma ATRIE

Implementação mais simples: R-way

- A árvore contém dois tipos de nós:
  - Nó interno: 'I'
  - Nó de informação / palavra: 'P'
- Cada nó contém todos os valores do alfabeto.
  (há desperdício de espaço)

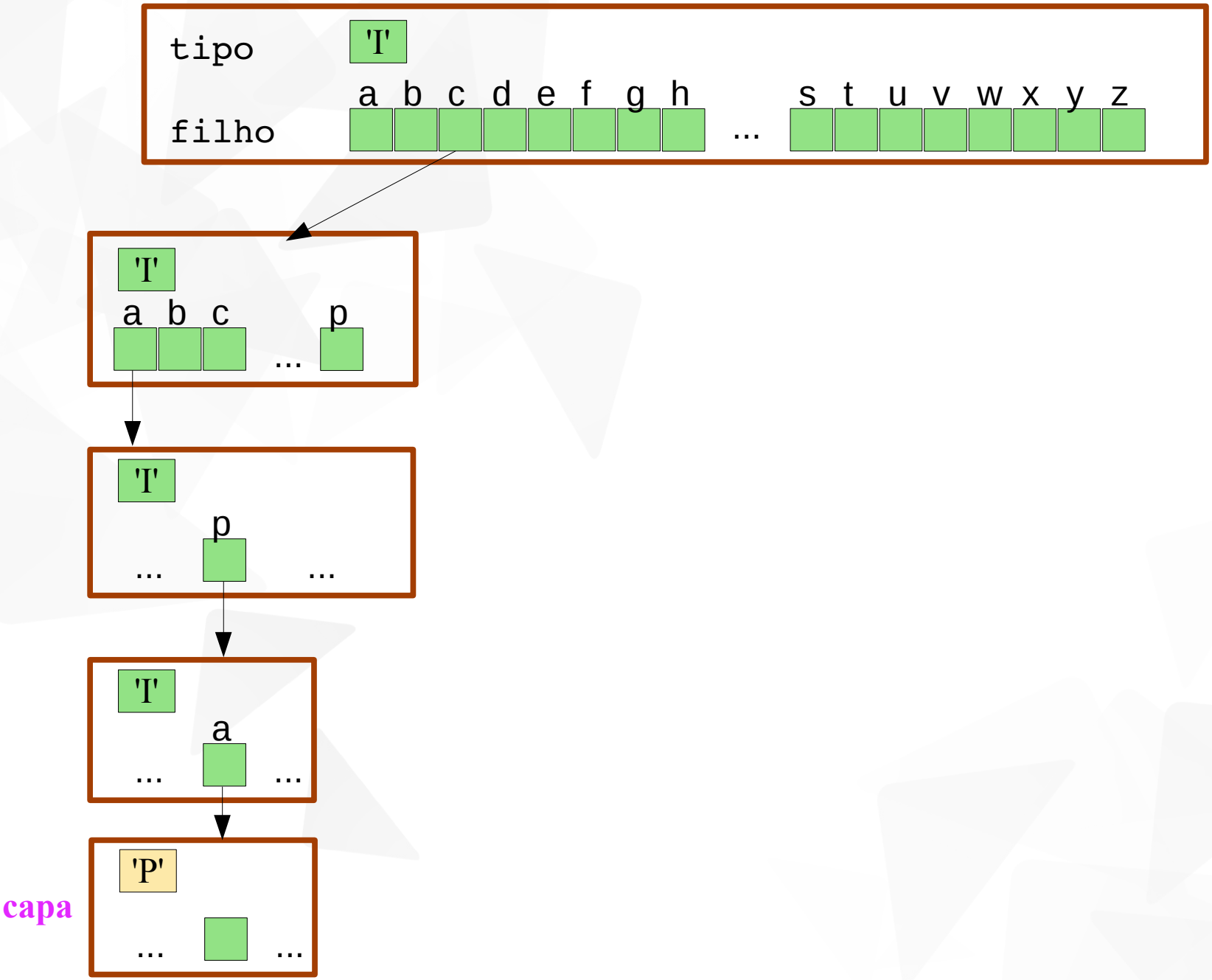
Considere uma ATRIE para armazenar chaves do alfabeto {a,b,c,d,...,y,z} (27 letras)

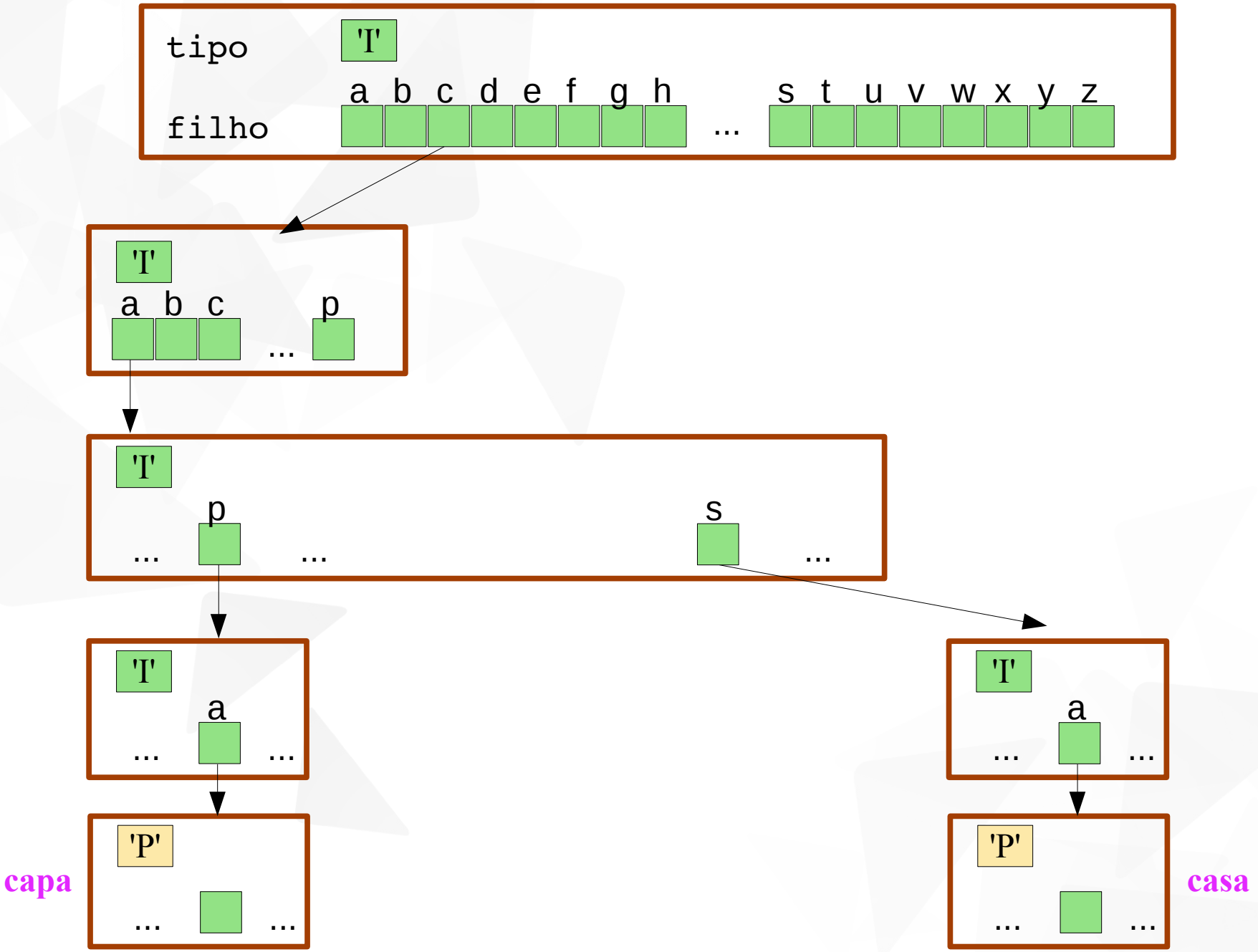
Implementando uma ATRIE

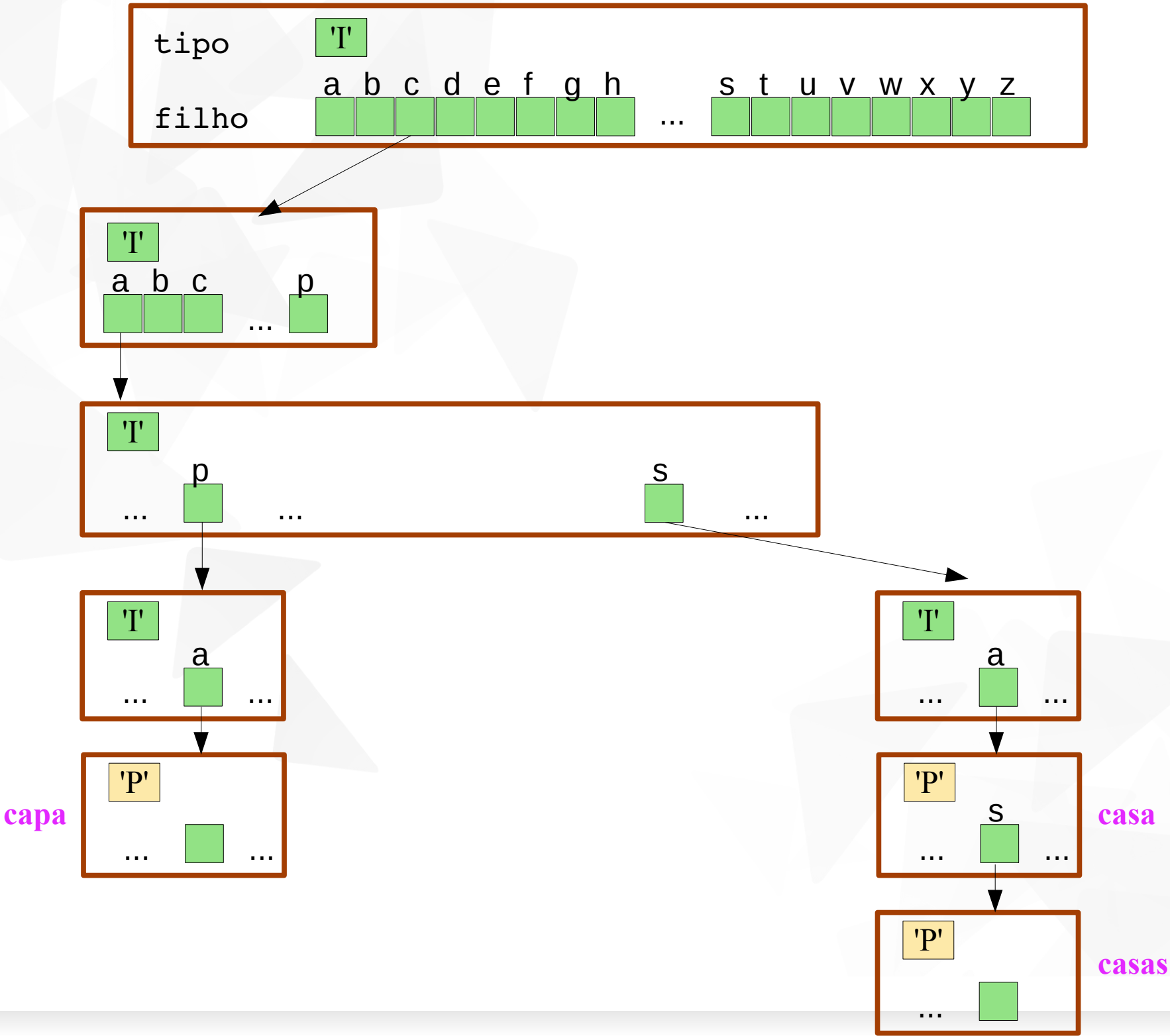
```c
#define TAMANHO_ALFABETO (27)
#define CHAR_TO_INDEX(c) ((int)c - (int)'a')
//#define CHAR_TO_INDEX(c) ((int)c - 97)
```

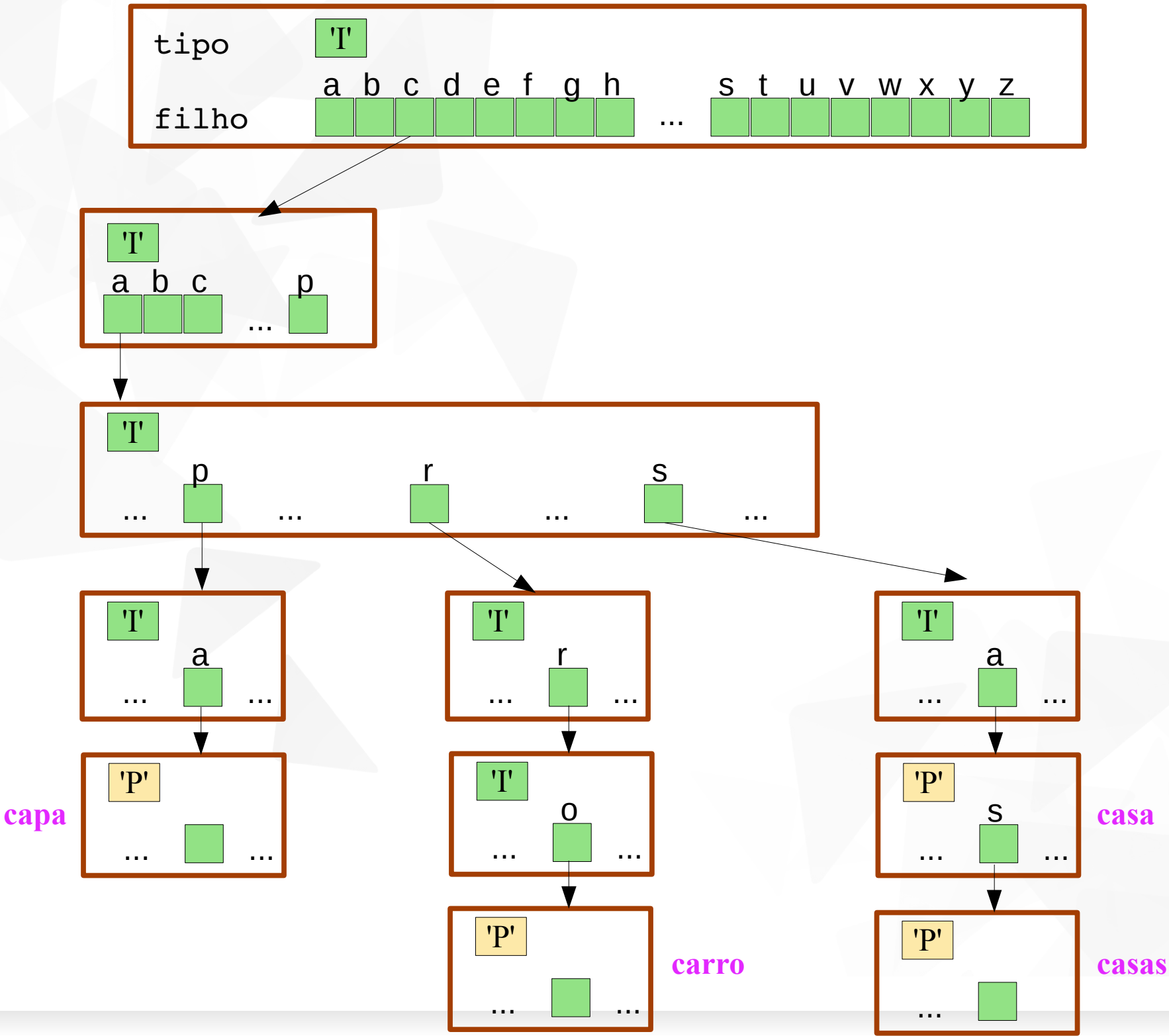
```c
struct trie_cel {
    char tipo; // 'I': interno / 'P': palavra
    struct trie_cel *filho[TAMANHO_ALFABETO];
};


typedef struct trie_cel no;
```

```c
no *criarNo(void) {
    int i;
    no *novo = (no *)malloc(sizeof(no));

    if( novo != NULL ) {
        novo->tipo = 'I';

        for(i=0; i<TAMANHO_ALFABETO; i++)
            novo->filho[i] = NULL;
    }
    return novo;
}
```

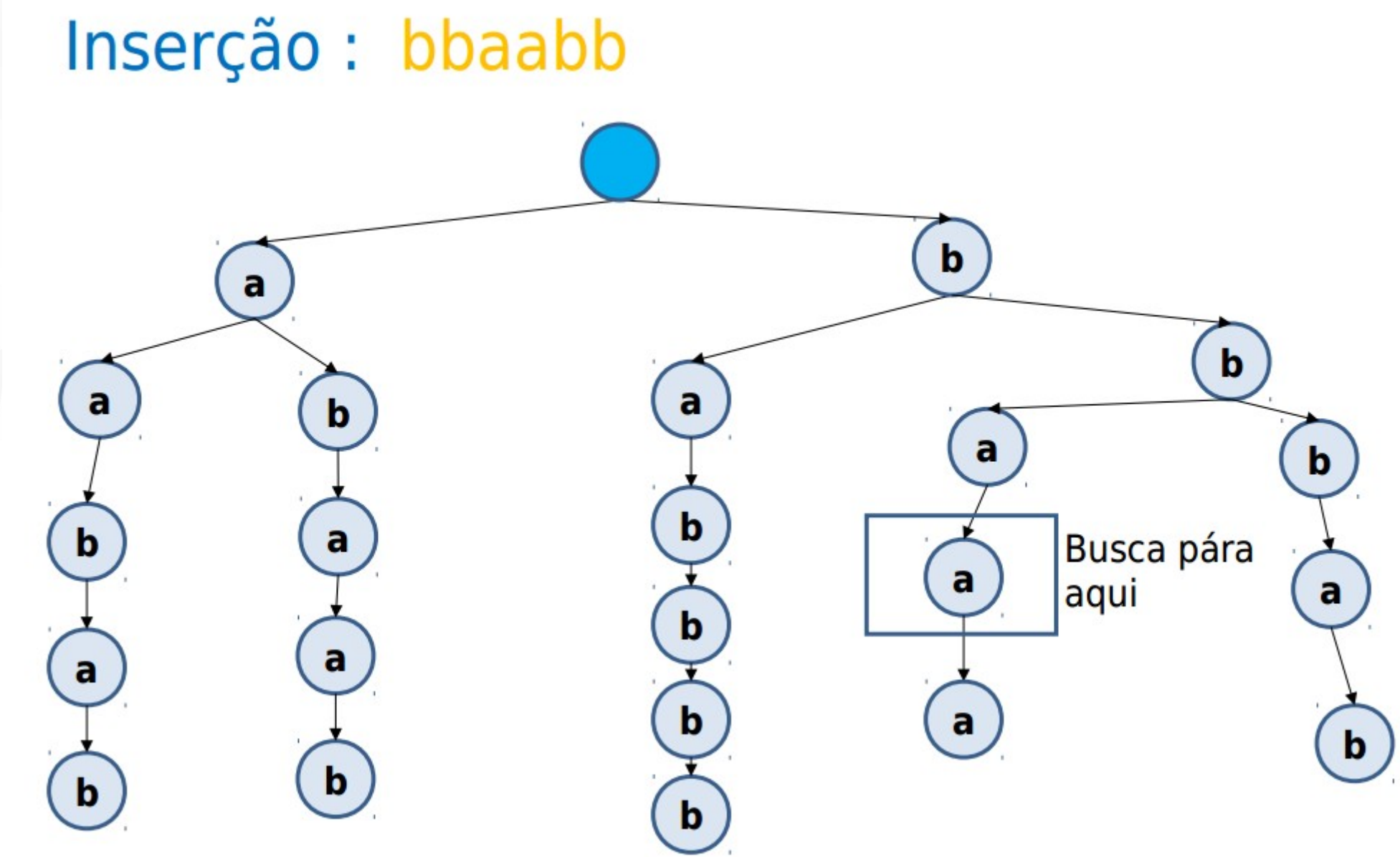

tipo
'I'
a b c d e f g h ... s t u v w x y z
filho
'I'
a b c ... p
'I'
p
... ...
'I'
a
... ...
'P'
... ...
capa


tipo
'I'
a b c d e f g h ... s t u v w x y z
filho
'I'
a b c ... p
'I'
p ... ... s ...
'I'
a ... ...
'P'
... ...
capa
'I'
a ... ...
'P'
... ...
casa

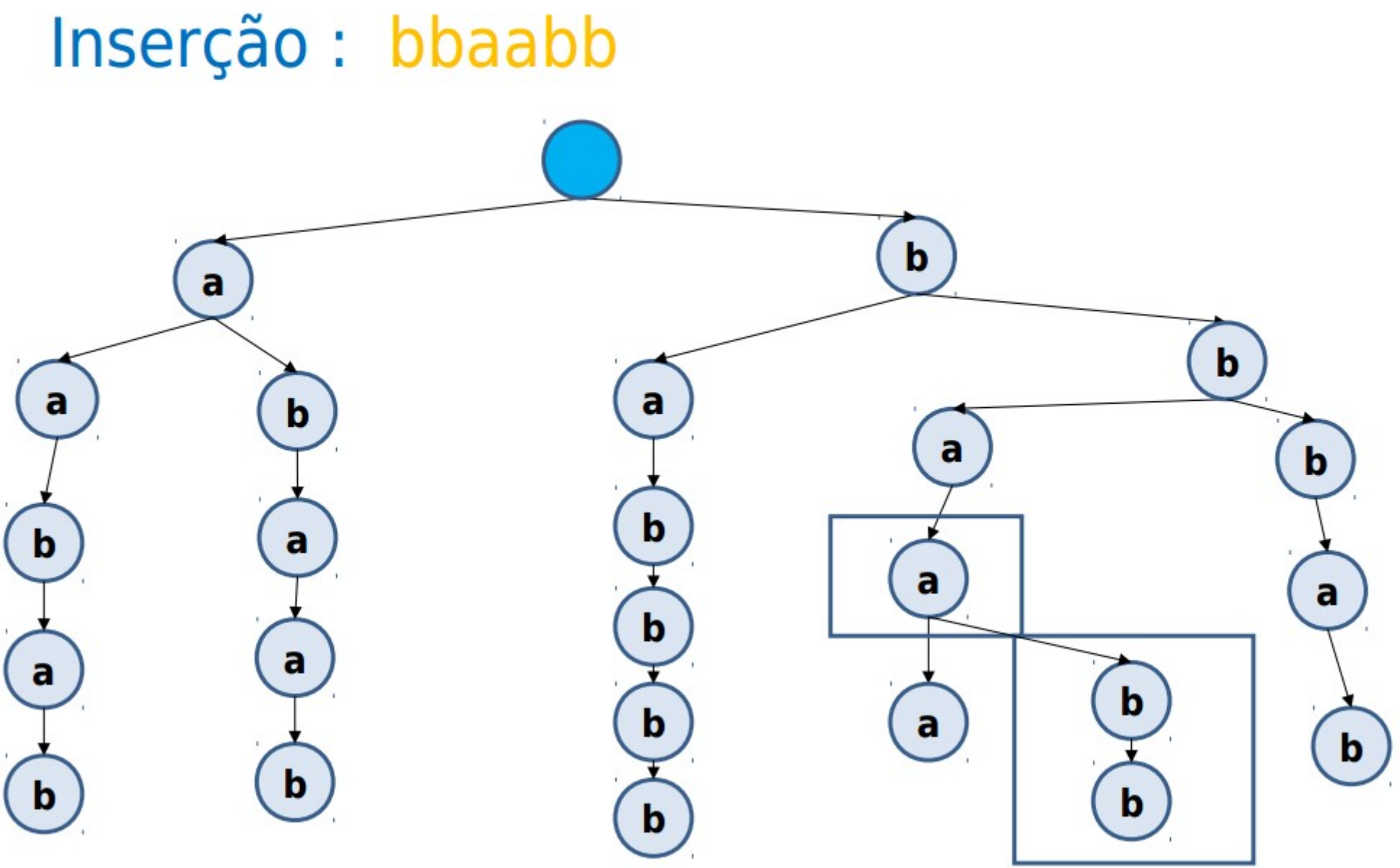

tipo
'I'
filho
a b c d e f g h ... s t u v w x y z
'I'
a b c ... p
'I'
p ... ... s ...
'I'
a ... ...
'P'
capa
... ...
'I'
a ... ...
'P'
s
casa
... ...
'P'
casas
...


tipo
'I'
a b c d e f g h ... s t u v w x y z
filho
'I'
a b c ... p
'I'
p ... ... r ... ... s ...
'I'
a ... ...
'P'
capa
... ...
'I'
r ... ...
'I'
o ... ...
'P'
carro
... ...
'I'
a ... ...
'P'
s ... ...
casa
'P'
casas
...

Inserção em uma ATRIE



Inserção em uma ATRIE



Inserção em uma ATRIE

```
#define TAMANHO_ALFABETO (27)
#define CHAR_TO_INDEX(c) ((int)c - (int)'a')
//#define CHAR_TO_INDEX(c) ((int)c - 97)
```

```
void adicionarPalavra(char *palavra, no *raiz) {
    int nivel=0, indice;
    no *tmp = raiz;

    for(nivel=0; nivel<strlen(palavra); nivel++) {
        indice = CHAR_TO_INDEX(palavra[nivel]);
        if( tmp->filho[indice]==NULL )
            tmp->filho[indice] = criarNo();
        tmp = tmp->filho[indice];
    }
    tmp->tipo = 'P';
}
```

Busca em uma ATRIE

```
#define TAMANHO_ALFABETO (27)
#define CHAR_TO_INDEX(c) ((int)c - (int)'a')
//#define CHAR_TO_INDEX(c) ((int)c - 97)
```

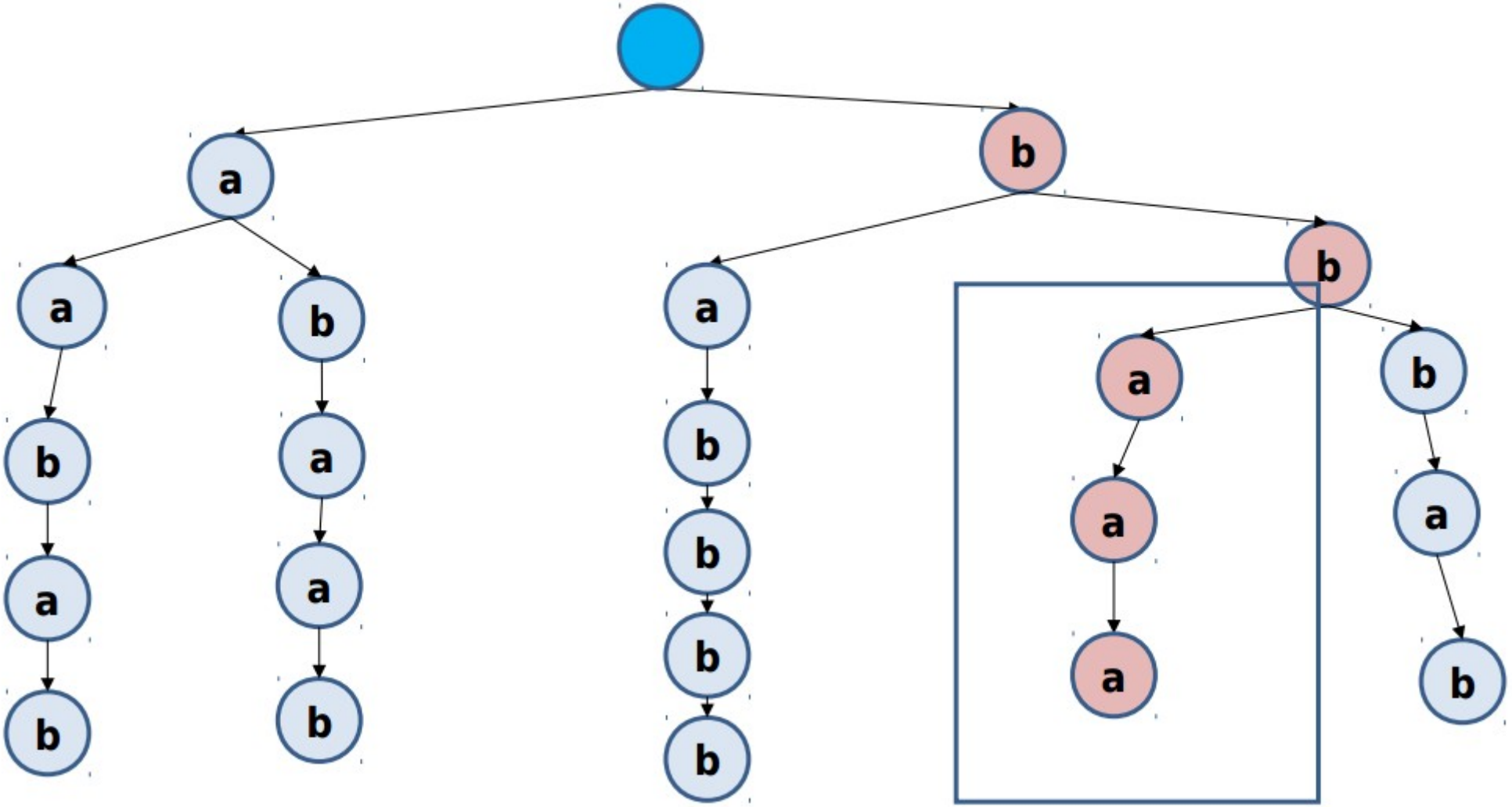
```
int buscaPalavra(char *palavra, no *raiz) {
    int i, indice;
    no *tmp = raiz;

    for(i=0; i<strlen(palavra); i++) {
        indice = CHAR_TO_INDEX(palavra[i]);

        if(tmp->filho[indice] != NULL)
            tmp = tmp->filho[indice];
        else
            break;
    }

    if(palavra[i]=='\0' && tmp->tipo=='P')
        return 1;
    else
        return 0; // palavra nao esta na arvore
}
```

Remoção em uma ATRIE

- **Busca-se o nó que representa o final da palavra a ser removida.**
- **São removidos os nós que possuem apenas um filho pelo caminho ascendente.**
- **A remoção é concluída quando se encontra um nó com mais de um filho.**

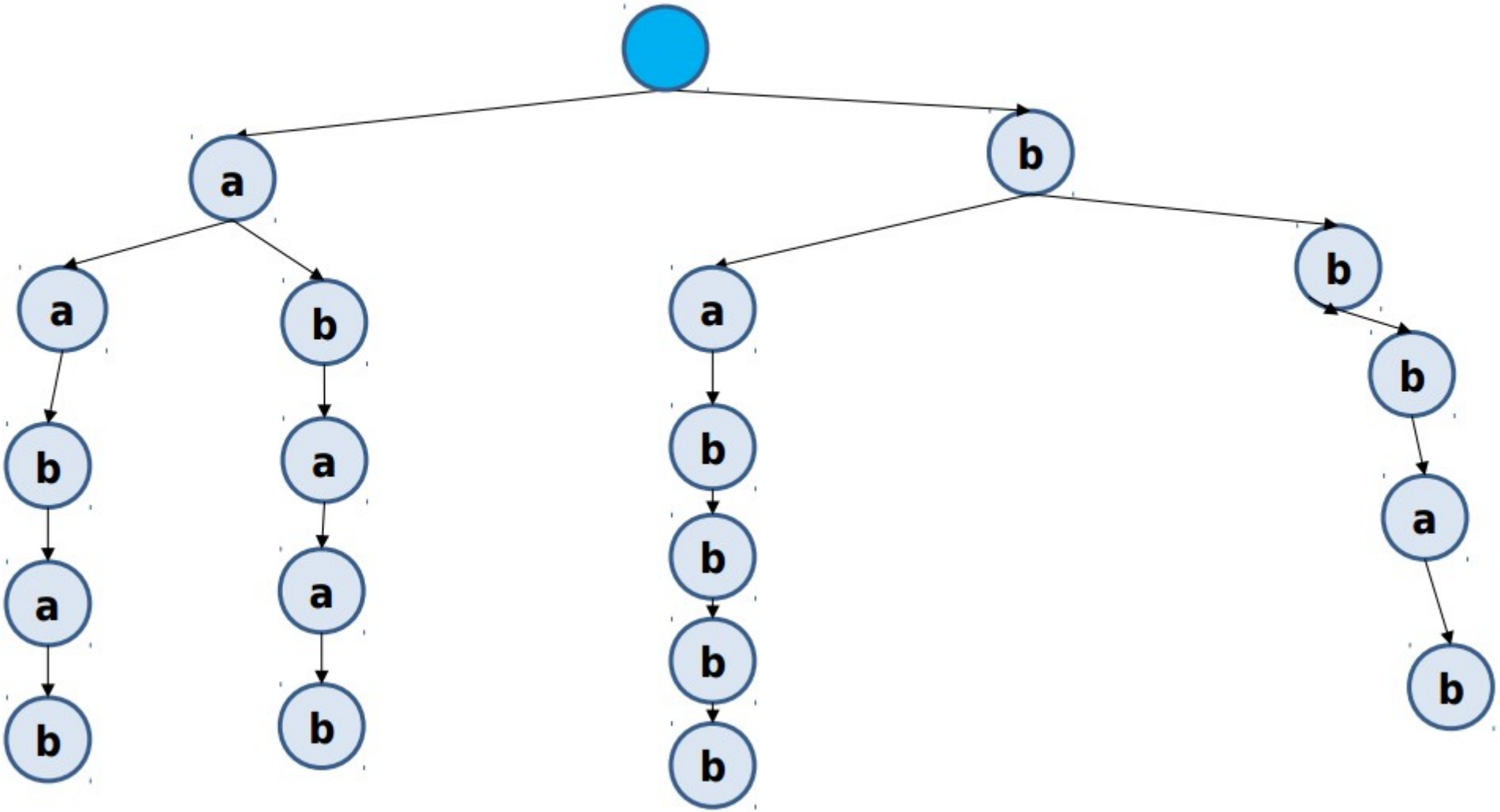
Remoção em uma ATRIE

Remoção : bbaaa



Remoção em uma ATRIE

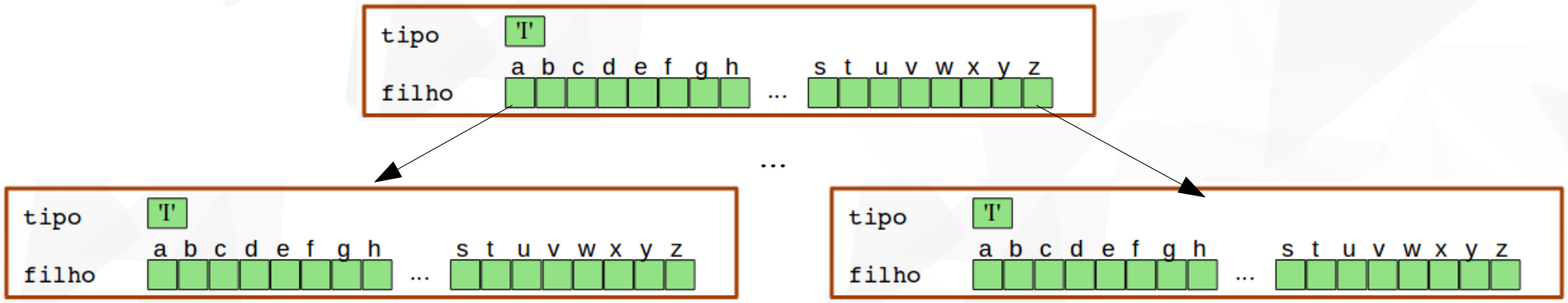
Remoção : bbaaa



Complexida computacional

- **Busca, Inserção e Remoção:**
  Para uma palavra p de tamanho |p|: O(|p|)
  Independe do número total de palavras.

- **Uso de memória:**
  Ineficiente para armazenar todas as palavras.



Aplicações

Árvores TRIE: Aplicações

Busca por aproximação de correspondência:

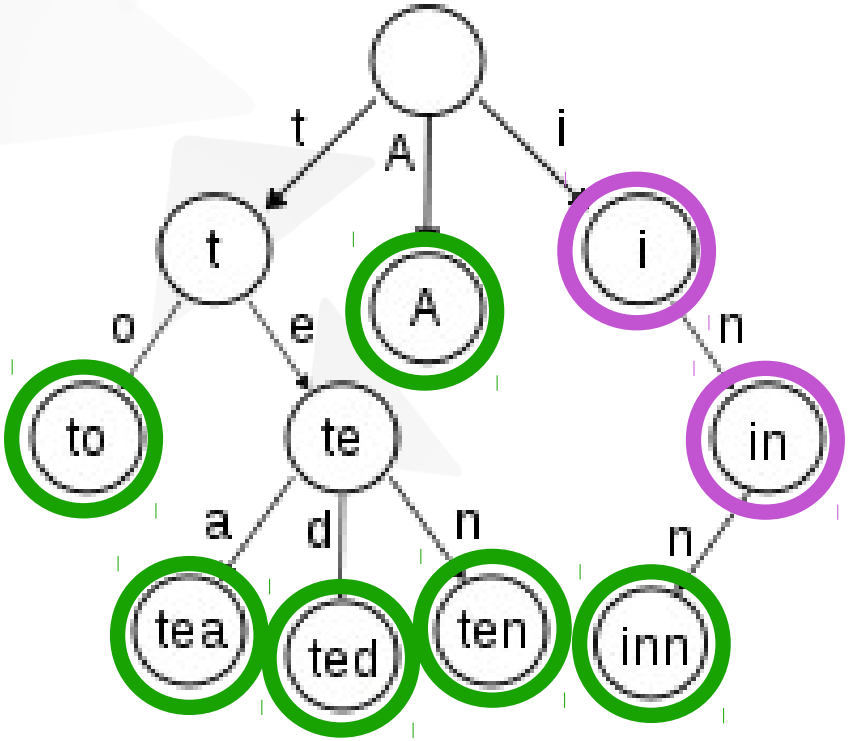
Onde podem ser localizadas dados que são semelhantes a uma chave informada.

- Corretor Ortográfico
- Auto-preenchimento

Árvores TRIE: Aplicações

Corretor ortográfico

Nesse tipo de programas, as palavras são comparadas com um dicionário armazenado em arquivo, e se não são encontradas, indica-se as opções de correção.

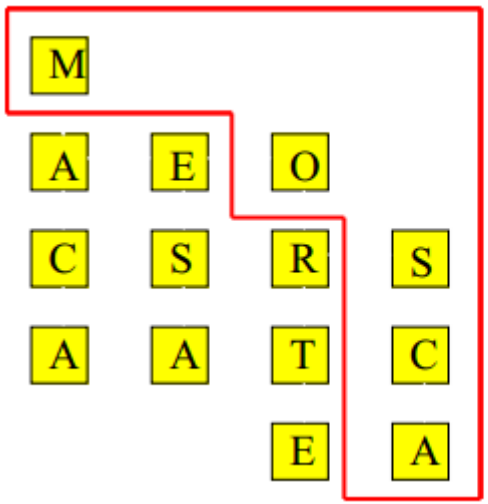


Palavra a ser comparada/procurada: **tex**

Árvores TRIE: Aplicações

Auto-preenchimento

Nesse tipo de programas, a medida que vai digitando são exibidas as opções possíves de palavras já utilizadas.



M O S C A

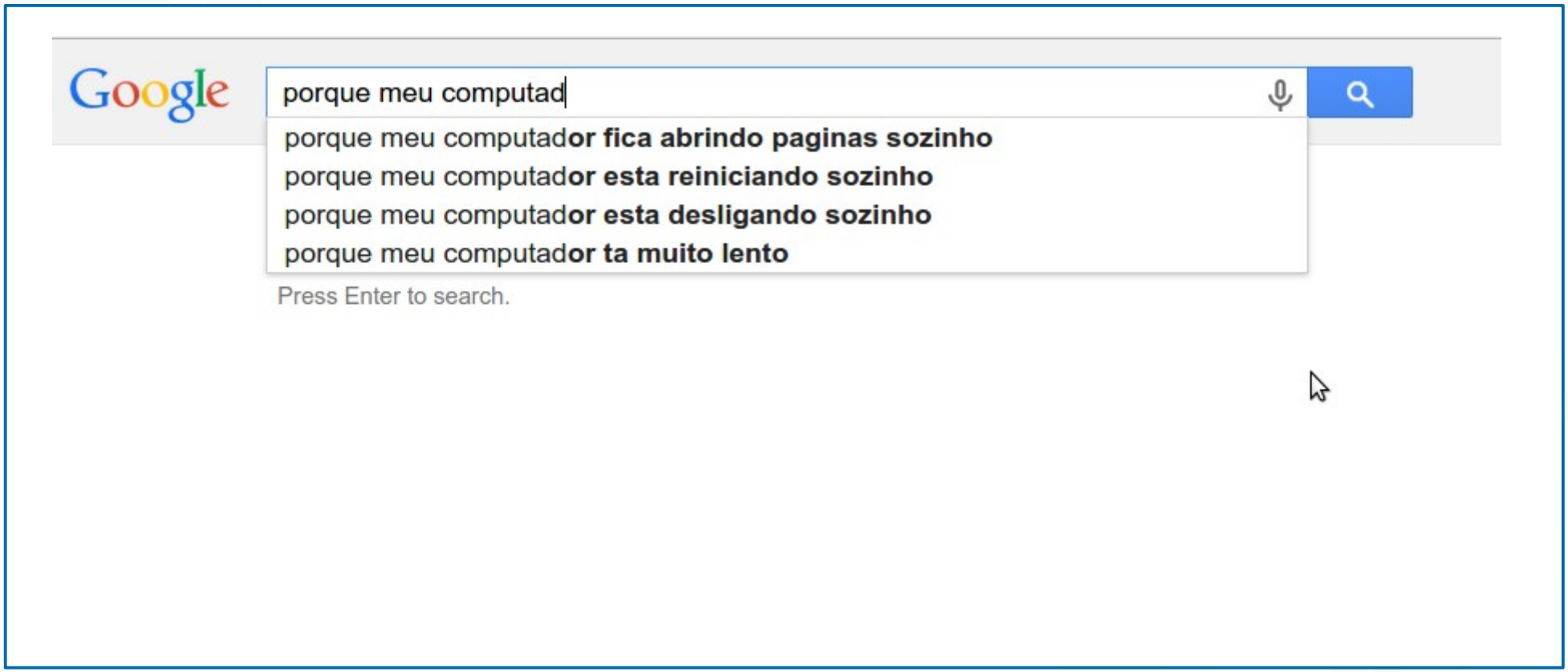
O algoritmo compara a existência de correspondência na estrutura.

A cada caractere digitado, são apresentadas as opções de preenchimento, e no momento em que só existir um caminho possível a ser seguido na TRIE ocorre o preenchimento automático

Árvores TRIE: Aplicações

Auto-preenchimento

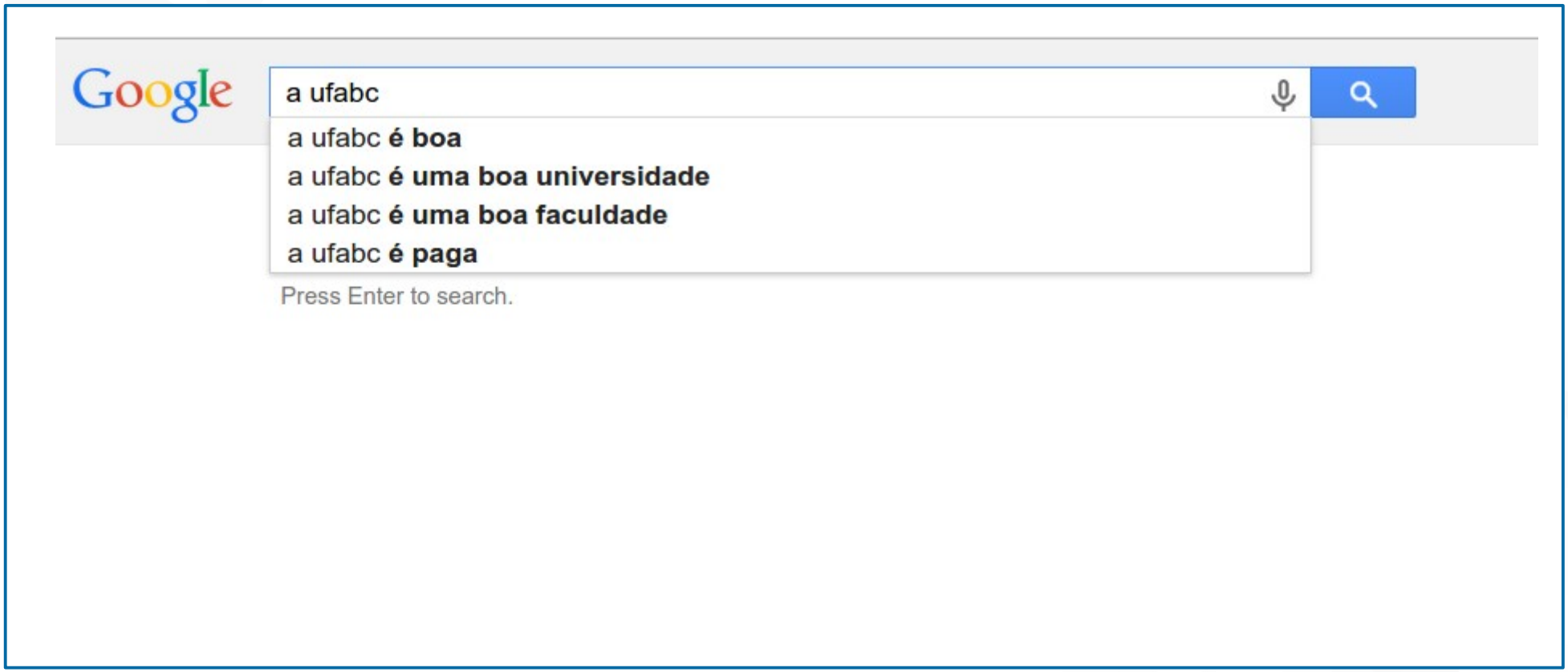
Nesse tipo de programas, a medida que vai digitando são exibidas as opções possíves de palavras já utilizadas.

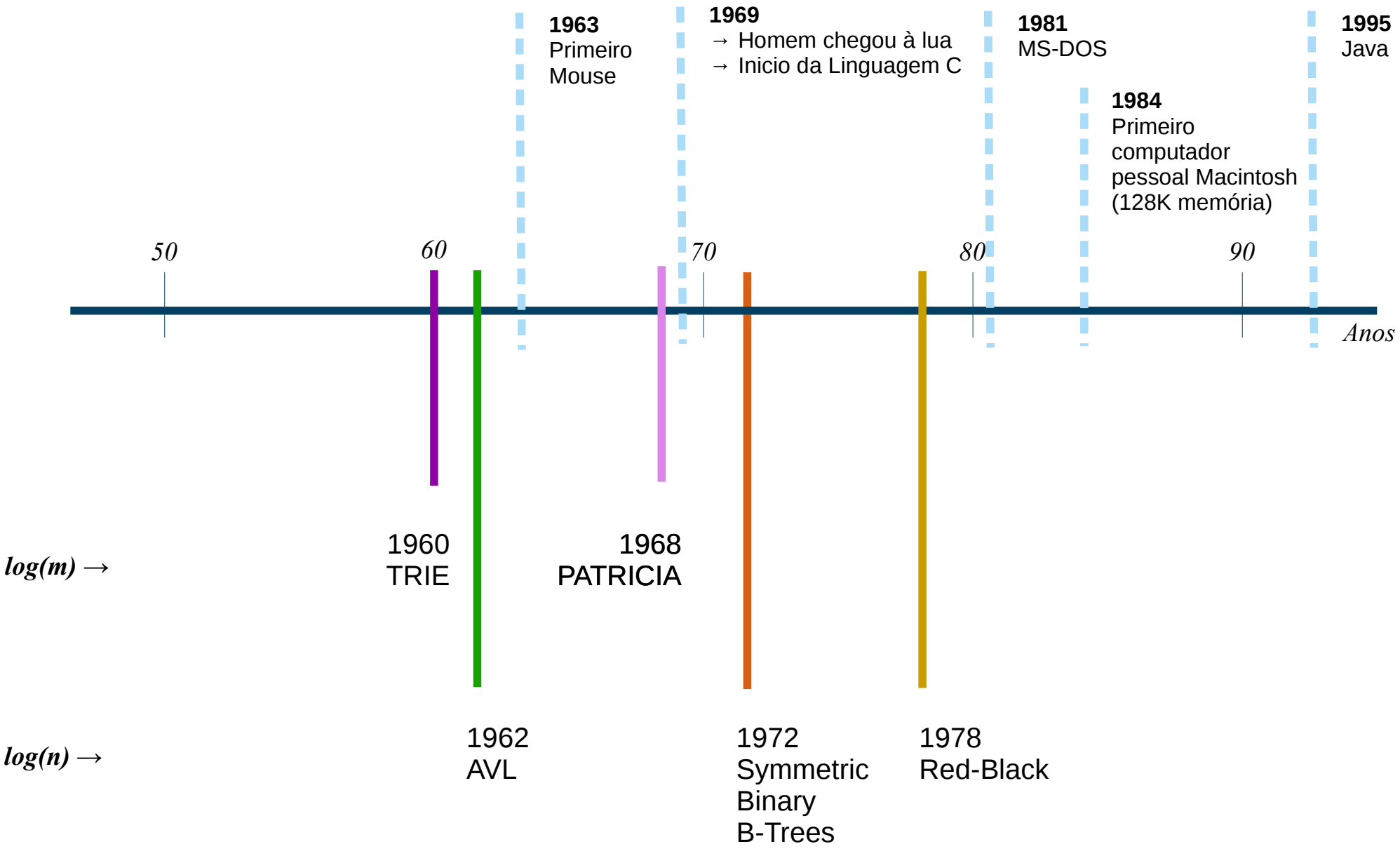


Árvores TRIE: Aplicações

Auto-preenchimento

Nesse tipo de programas, a medida que vai digitando são exibidas as opções possíves de palavras já utilizadas.




1963
Primeiro
Mouse
1969
→ Homem chegou à lua
→ Inicio da Linguagem C
1981
MS-DOS
1984
Primeiro
computador
pessoal Macintosh
(128K memória)
1995
Java
50
60
70
80
90
Anos
log(m) →
1960
TRIE
1968
PATRICIA
log(n) →
1962
AVL
1972
Symmetric
Binary
B-Trees
1978
Red-Black