Árvores Digitais

Fonte de consulta: Szwarcfiter, J.; Markezon, L. Estruturas de Dados e seus Algoritmos, 3a. ed. LTC. Capítulo11

Premissas do que vimos até aqui

- As chaves têm tamanho fixo
- As chaves cabem em uma palavra do computador (para propiciar manipulação eficiente em memória)

Premissas do que vimos até aqui

- As chaves têm tamanho fixo
- As chaves cabem em uma palavra de computador (para propiciar manipulação eficiente em memória)

- Na prática, isso nem sempre acontece
  - Exemplo: aplicação que armazena um texto e permite busca por frases nesse texto
  - Chaves: frases
    - Tamanho variável
    - Não cabem em uma palavra de computador

Soluções que vimos até agora **não são aplicáveis** para indexar esse tipo de chave

- Na prática, isso nem sempre acontece
  - Exemplo: aplicação que armazena um texto e permite busca por frases nesse texto
  - Chaves: frases
    - Tamanho variável
    - Não cabem em uma palavra de computador

Solução

- Uso de Busca Digital
  - Árvore Digital
  - Árvore Digital Binária
  - Árvore Patrícia (implementação eficiente de Árvore Digital Binária)

Busca Digital

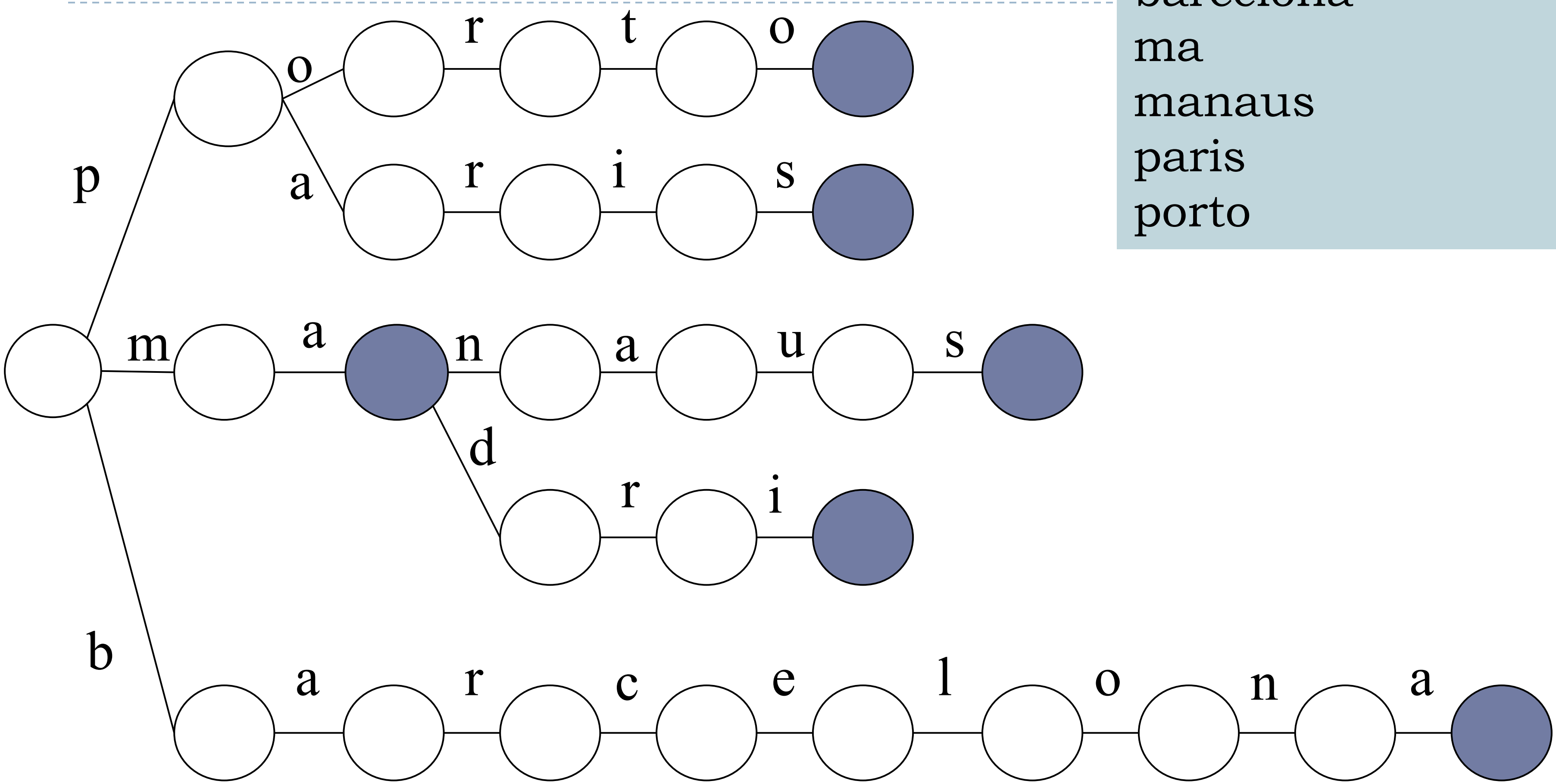
- **Árvore Digital**
- Árvore Digital Binária
- Árvore Patrícia (implementação eficiente de Árvore Digital Binária)

Árvores Digitais

- Também chamadas de **Tries**
- Chave não é indivisível
  - Considera-se que uma chave é uma sequência de dígitos que podem ser usados na indexação
- Ao invés de comparar a chave inteira, a comparação é feita **dígito a dígito**

Exemplo

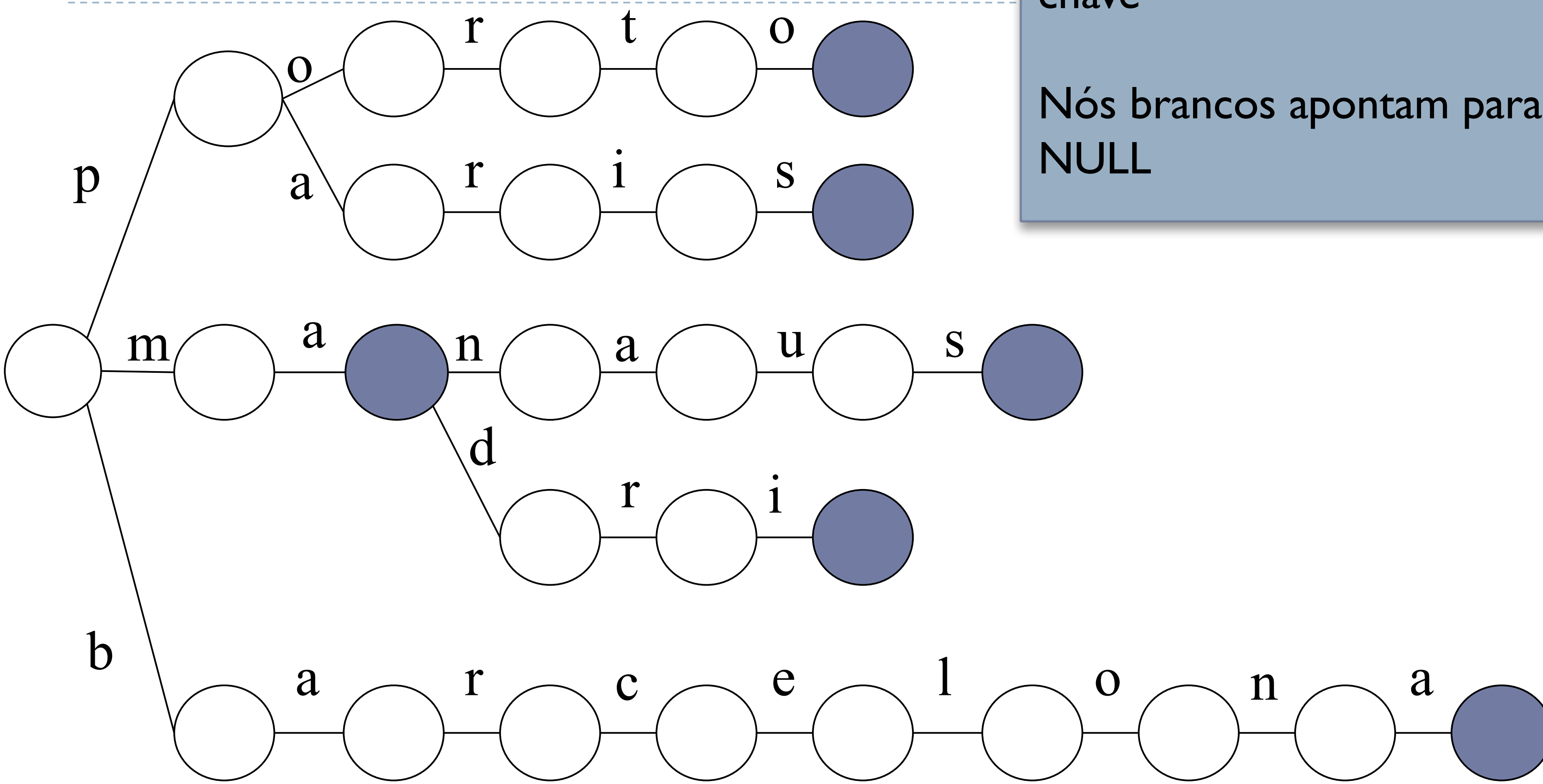
Chaves Indexadas:
madri
barcelona
ma
manaus
paris
porto



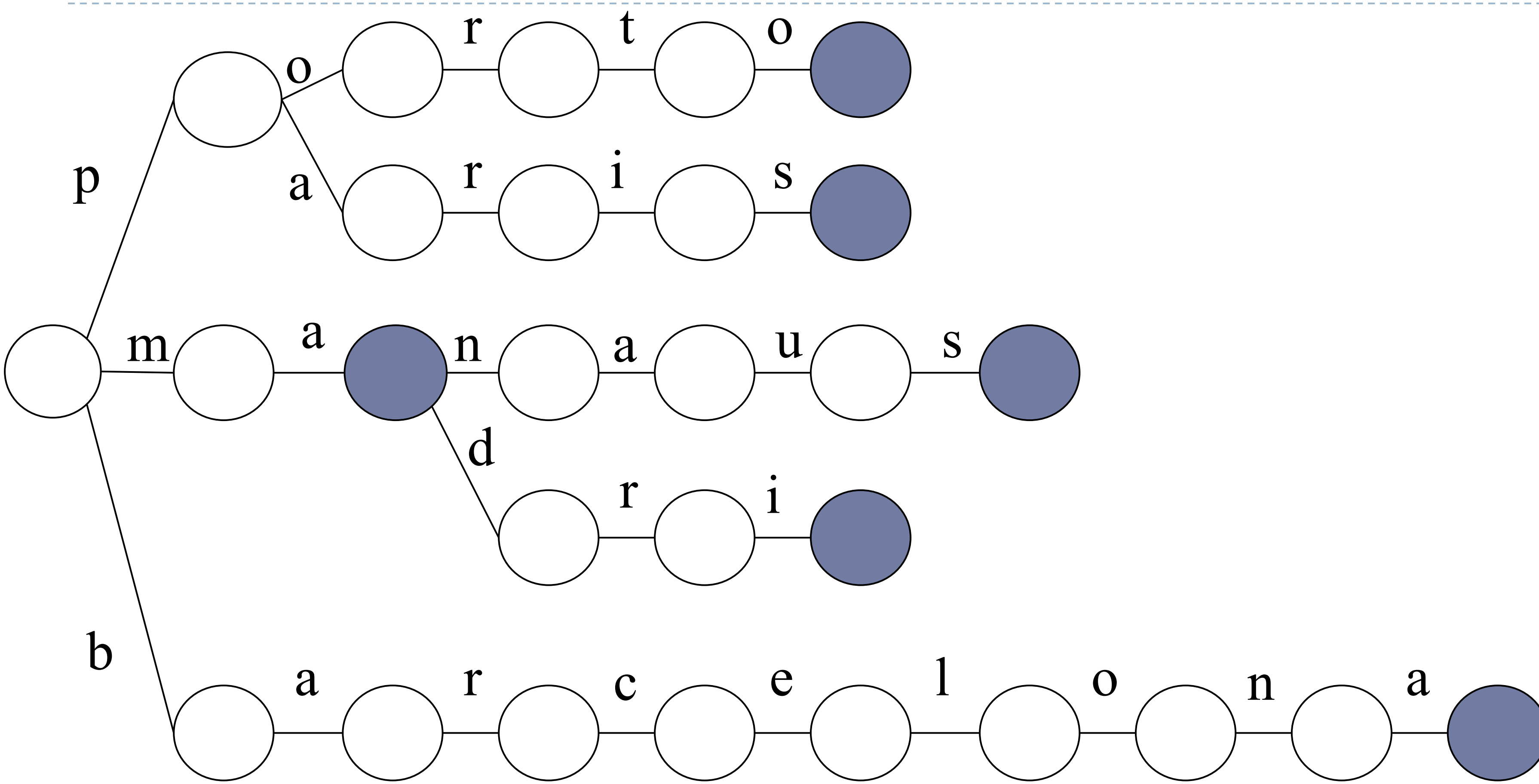
Exemplo

Nós azuis apontam para o registro que contém aquela chave

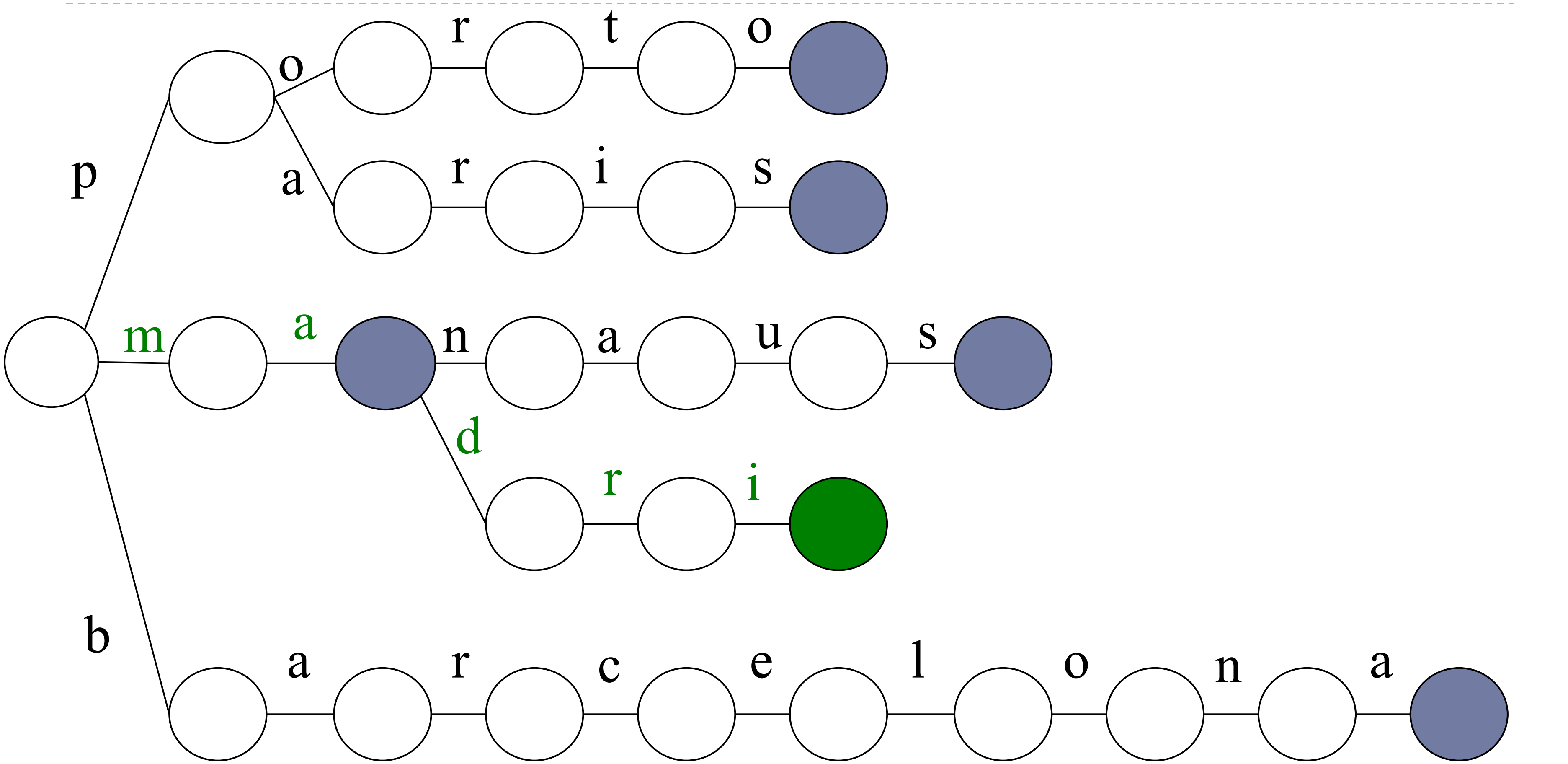
Nós brancos apontam para NULL



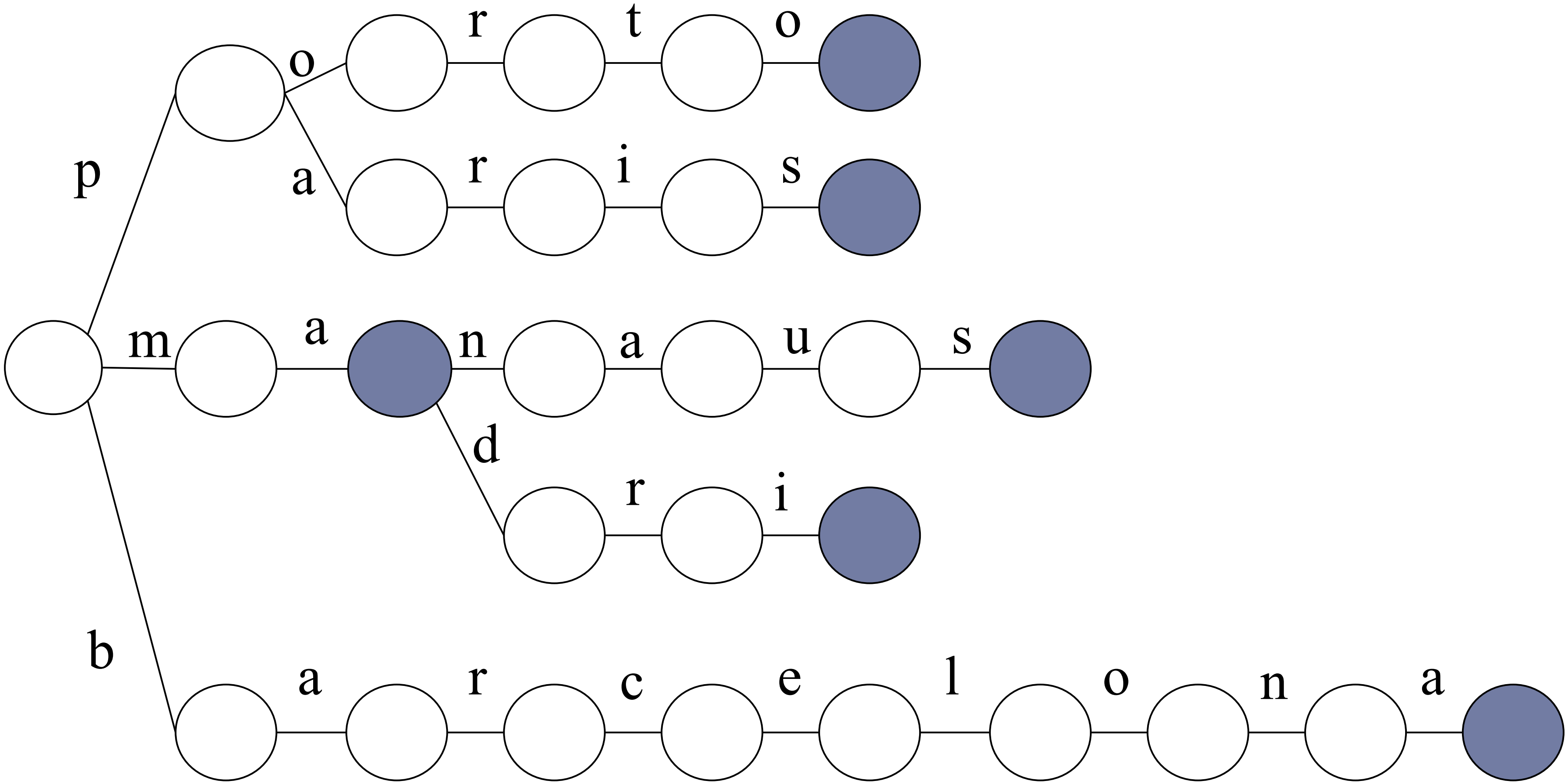
Exemplo: Buscar a chave madri



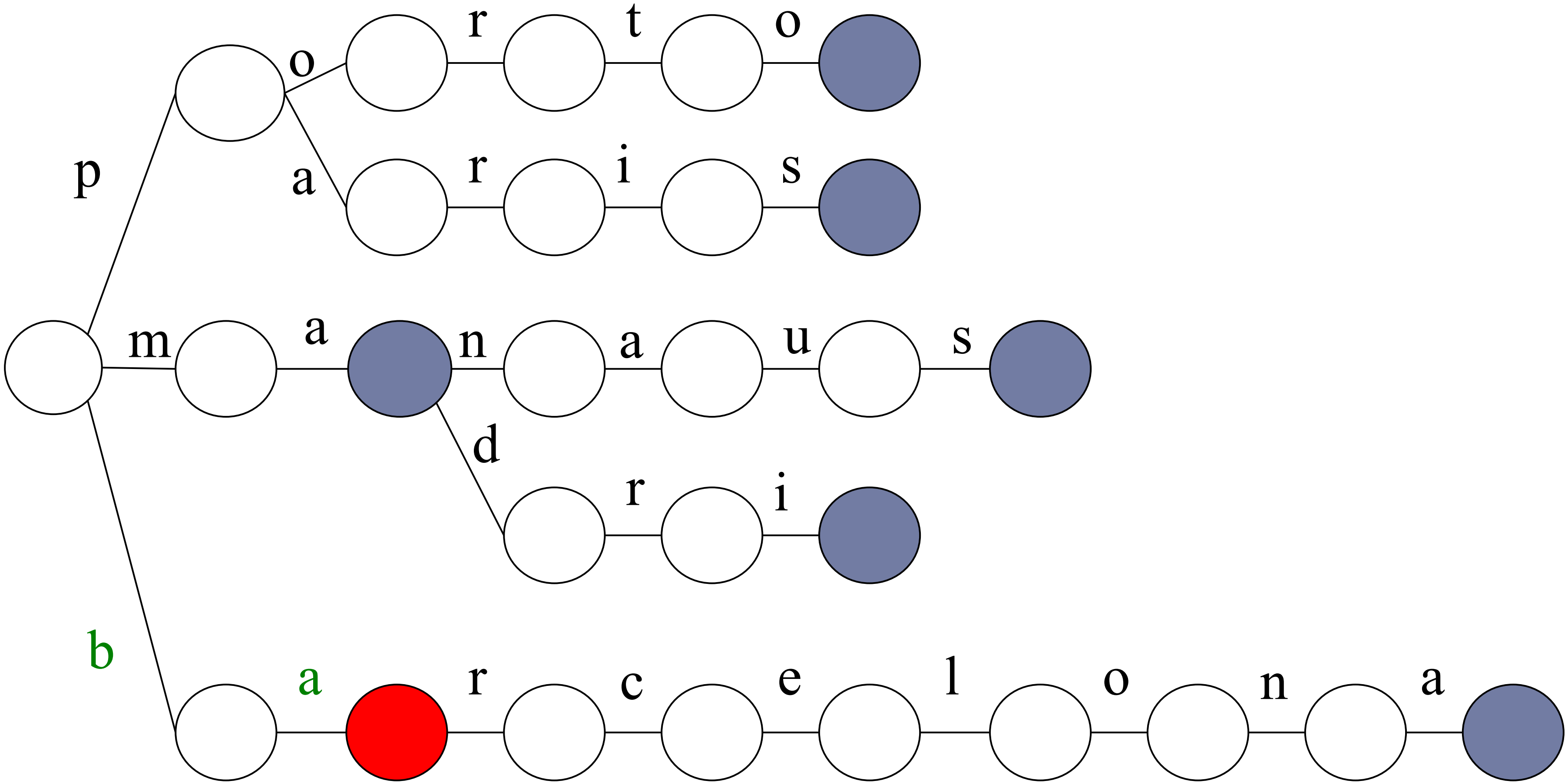
Exemplo: Buscar a chave madri



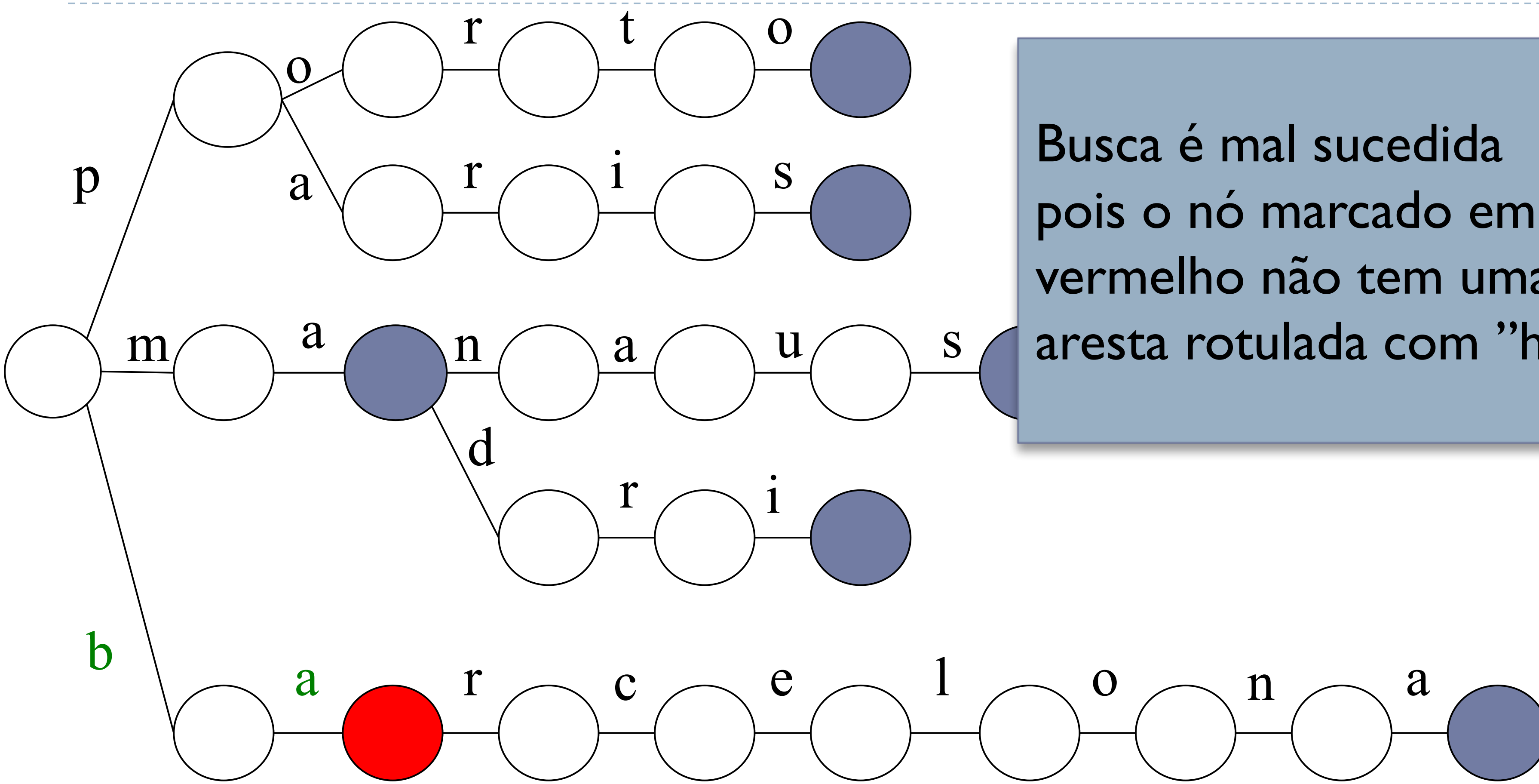
Exemplo: Buscar a chave bahia



Exemplo: Buscar a chave bahia



Exemplo: Buscar a chave bahia



Busca é mal sucedida pois o nó marcado em vermelho não tem uma aresta rotulada com ”h”

Definições

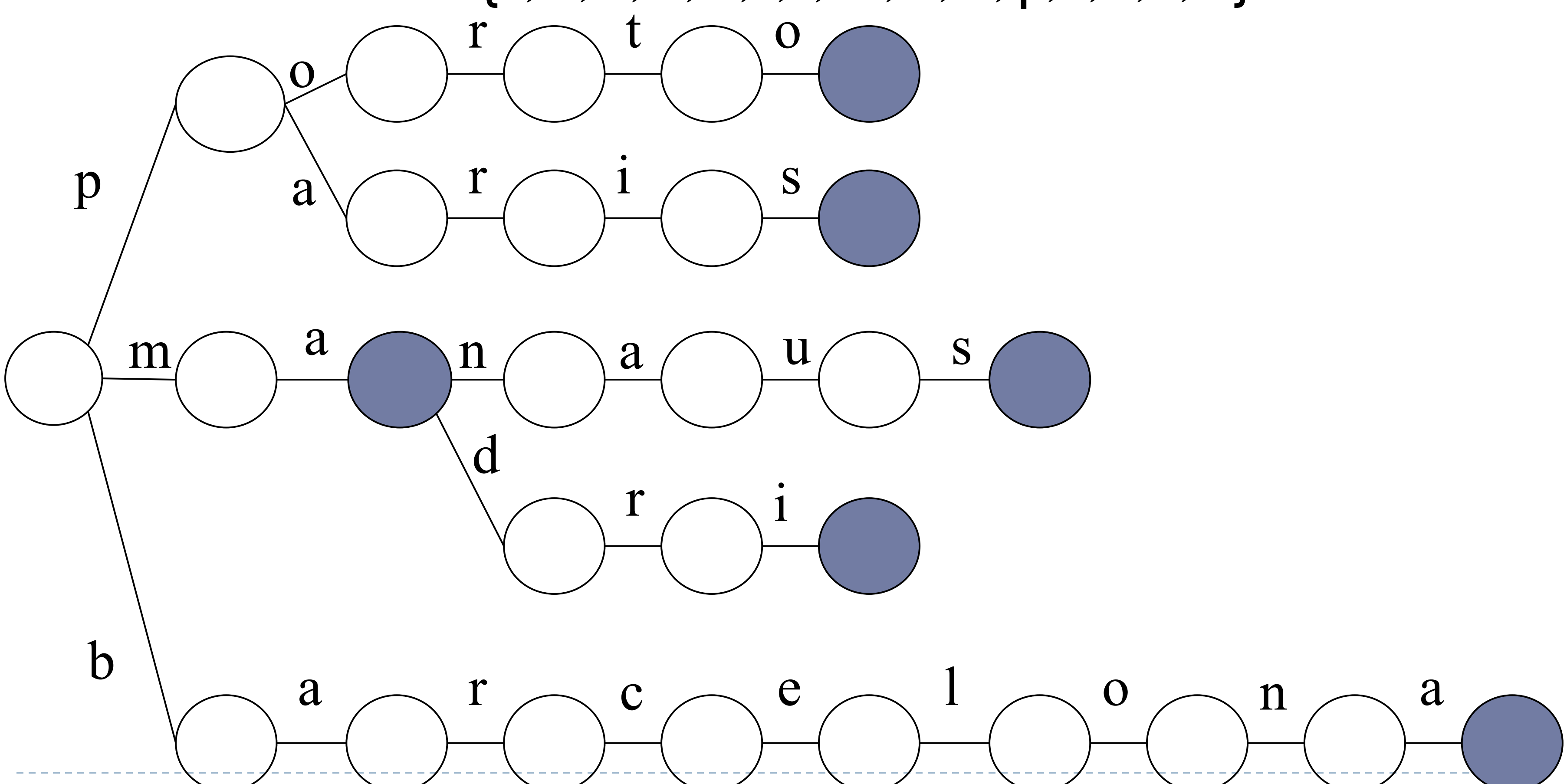
- $S = \{s_1, .., s_n\}$ é o conjunto de **chaves** a serem indexadas
- Cada chave s_i é formada por uma sequência de elementos d_j denominados **dígitos**
- Supõe-se que existe, em S, um total de m dígitos distintos, que compõe o **alfabeto** de S
- Os dígitos do alfabeto admitem ordenação, tal que $d_1 < ... < d_m$
- Os p primeiros dígitos de uma chave compõem o **prefixo de tamanho *p*** da chave

Definições

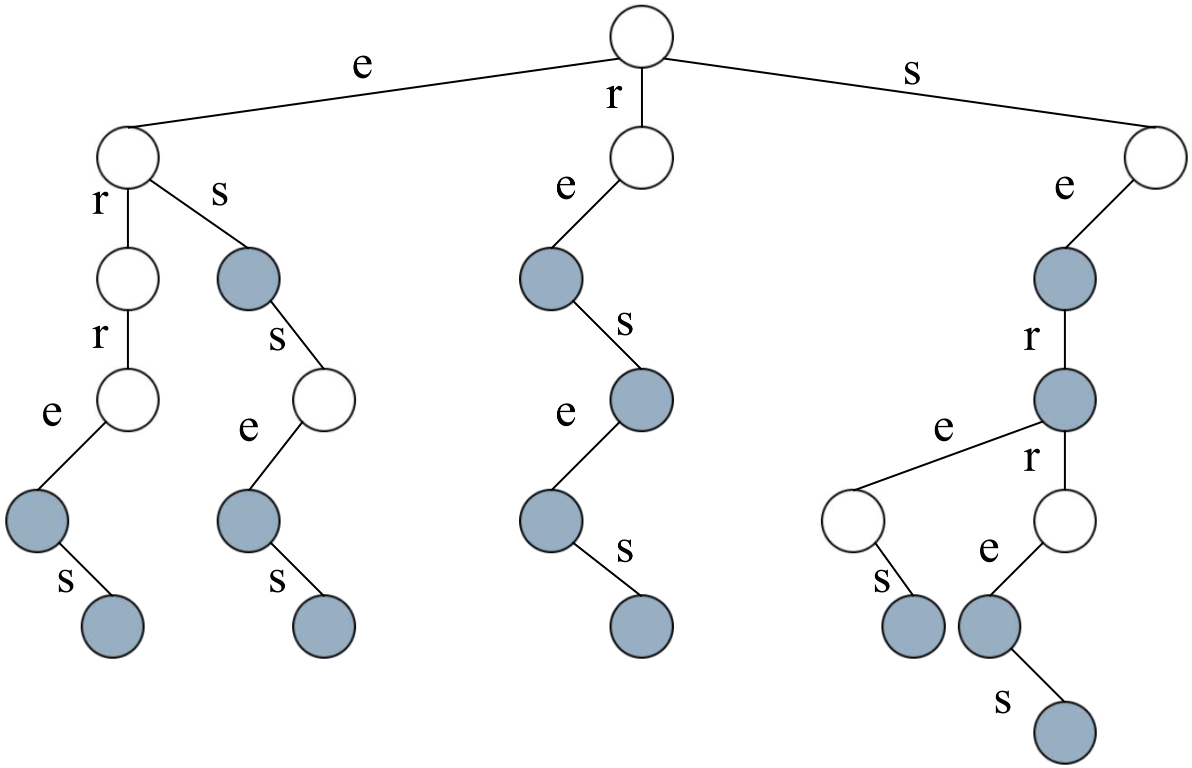
- Uma árvore digital para *S* é uma árvore *m*-ária *T*, não vazia, tal que:
  1. Se um nó *v* é o *j*-ésimo filho de seu pai, então *v* corresponde ao dígito d_j do alfabeto de *S* (isso exige que a posição dos nós que não existem seja preservada, para o caso de precisarem ser inseridos no futuro)
  2. Para cada nó *v*, a sequencia de dígitos definida pelo caminho desde a raiz de *T* até *v* corresponde a um prefixo de alguma chave de *S*
- A raiz da árvore sempre existe e não corresponde a nenhum dígito do alfabeto

No exemplo anterior

- S = {madri, barcelona, ma, manaus, paris, porto}
- Alfabeto de S = {a, b, c, d, e, i, l, m, n, o, p, r, s, t, u}



Exemplo que mostra espaço preservado



S = {erre, erres, es, esse, esses, se, ser, serre, re, res, rese, reses, serres, seres}
Alfabeto de S = {e, r, s} (árvore ternária)

Representação da árvore

- Alfabeto de tamanho m (árvore m-ária)
- Cada nó v apontado por pt ($pt \neq \lambda$) possui m filhos ordenados apontados por $pt\uparrow.pont[1]$, ..., $pt\uparrow.pont[m]$, respectivamente
- Se algum i-ésimo filho desse nó estiver ausente, então $pt\uparrow.pont[i] = \lambda$
- Se v for nó terminal de alguma chave, então $v.info$ = *terminal*. Caso contrário, $v.info$ = *não terminal*

Procedimento de Busca

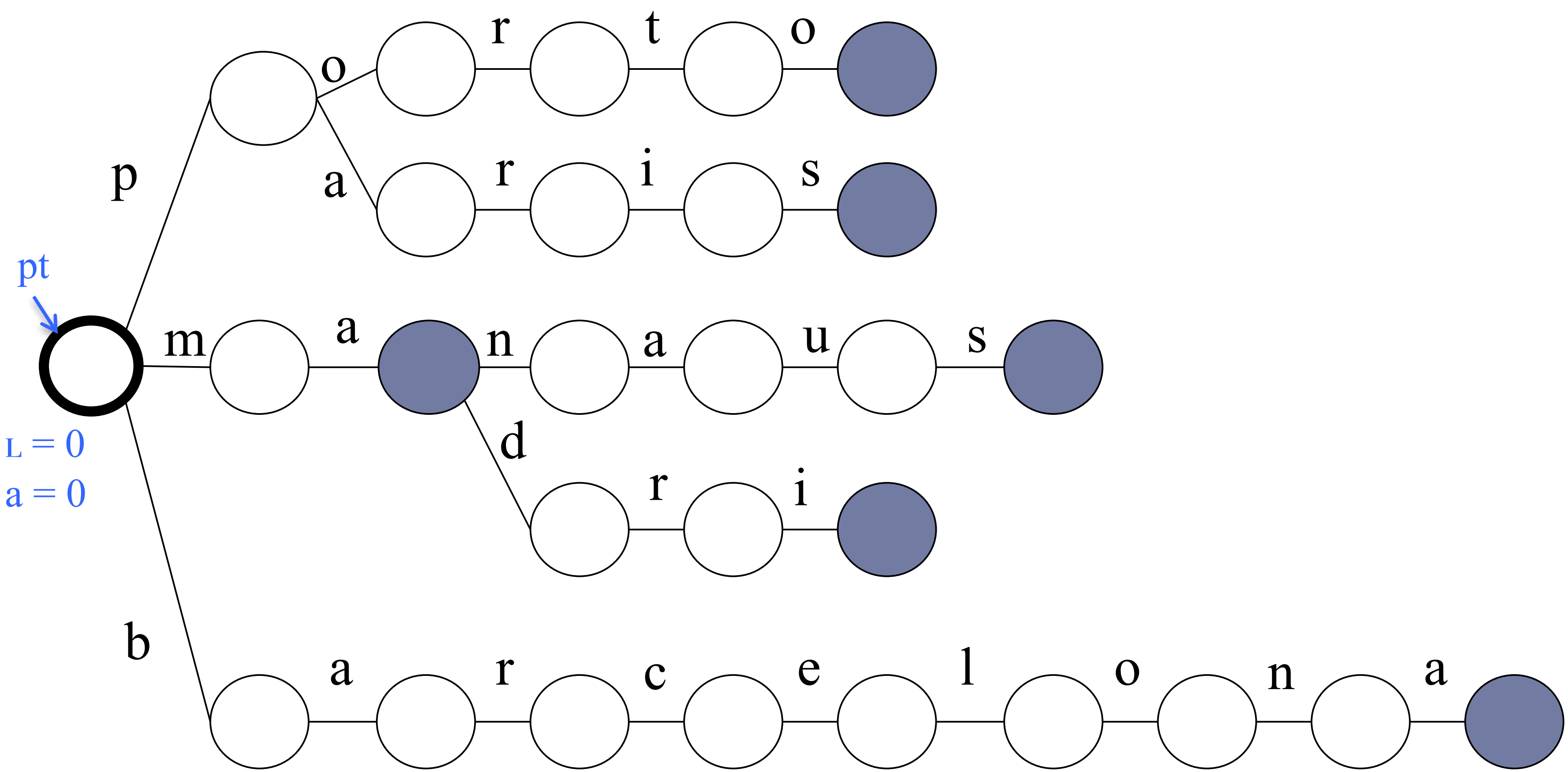
- A chave x a ser procurada possui k dígitos, denotados por d[1], ..., d[k]
- O parâmetro pt indica o nó corrente da árvore
- Os parâmetros L e **a** indicam o resultado da busca
  - **L** = tamanho do maior prefixo de x que coincide com um prefixo de alguma chave. Esse prefixo é obtido percorrendo-se a árvore desde sua raiz até o nó **w** apontado por **pt** ao final do processo
  - Se a = 1, chave foi encontrada no nó **w**, caso contrário, a = 0

Procedimento Busca Digital

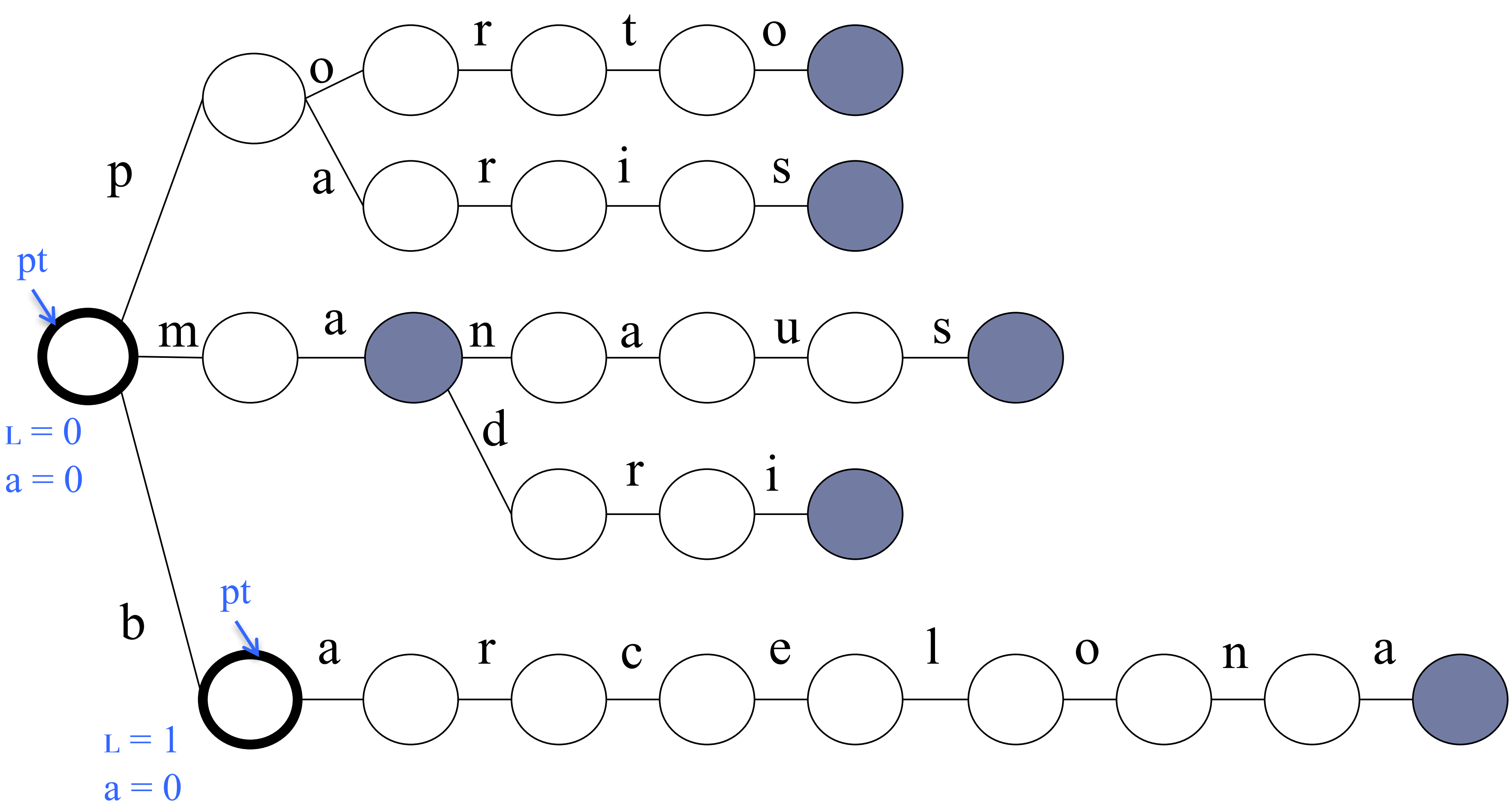
```
/* Procedimento assume que a árvore é
    representada conforme slides anteriores
   A chamada externa é:
 buscadig(x, pt, L, a), com pt = ptraiz, L = 0,
    a = 0; x é a chave a ser buscada
*/
procedimento buscadig(x, pt, l, a)
   se L < k então /* k é o número de dígitos da chave x */
      seja j a posição de d(L+1) na ordenação do alfabeto
      se pt↑.pont[j] ≠ λ então
         pt := pt↑.pont[j];
         L := L + 1
         buscadig(x, pt, L, a)
   senão se pt↑.info = terminal então a := 1
```


Fonte: Algoritmo 11.1, página 261

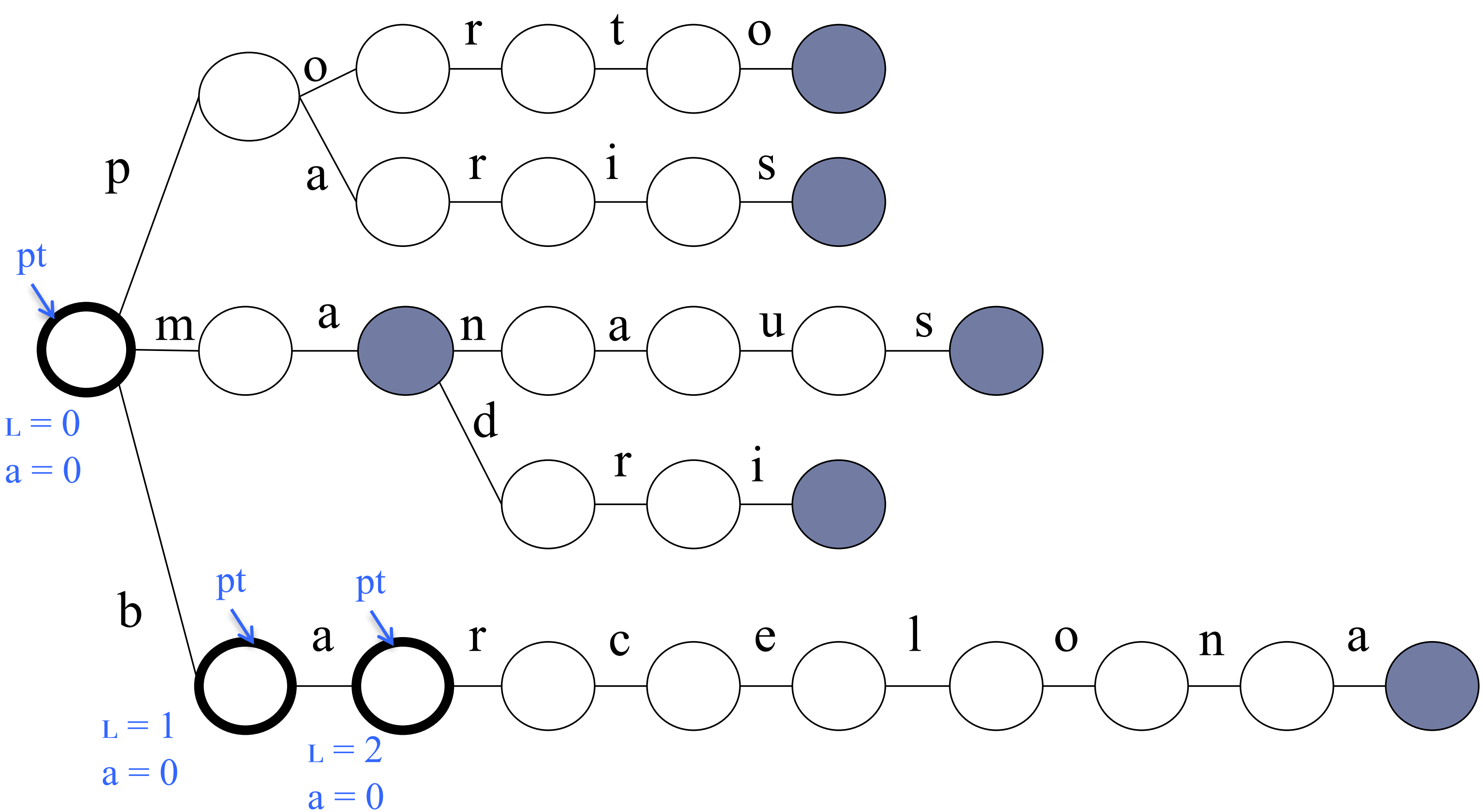
Exemplo: buscar chave bahia



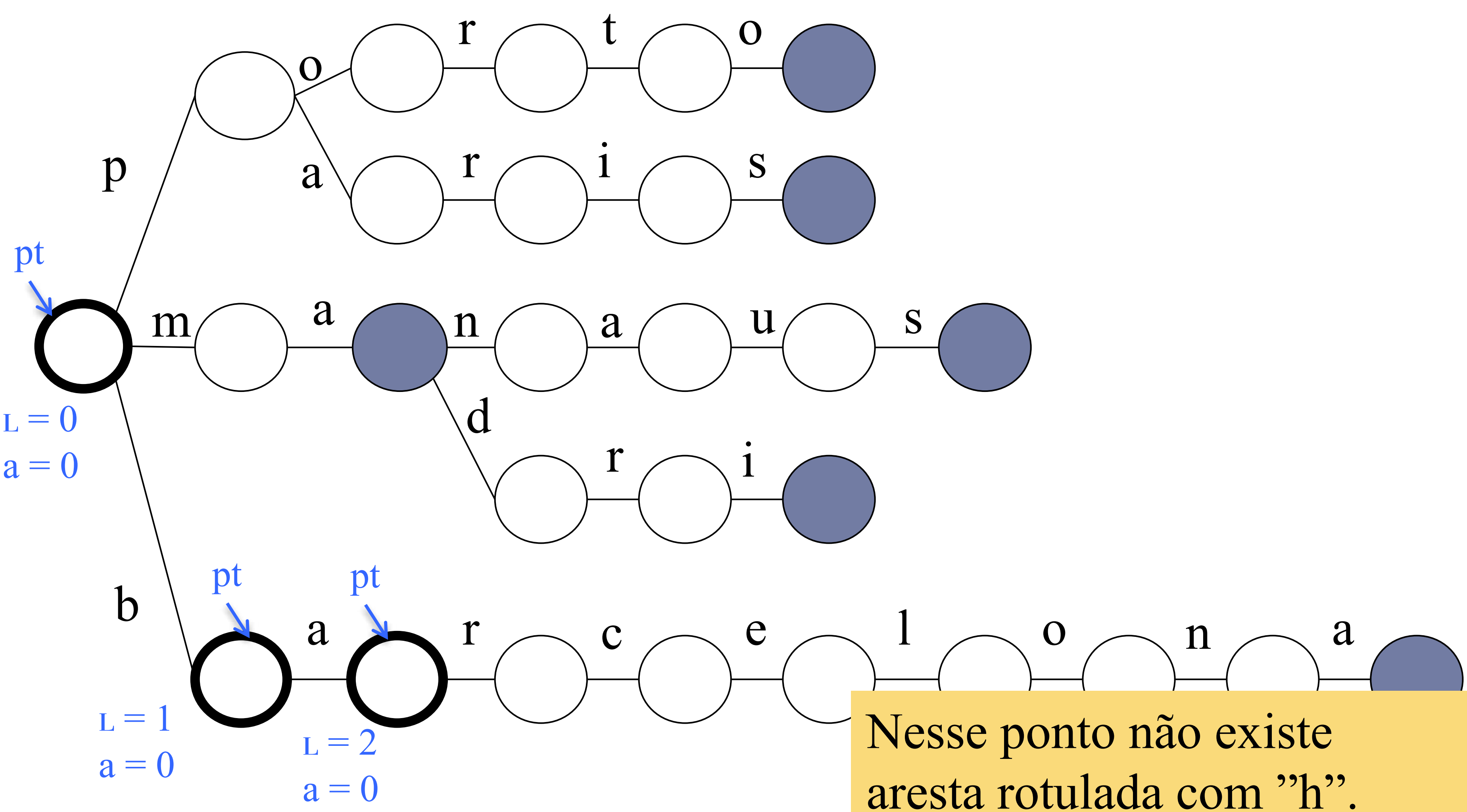
Exemplo: buscar chave bahia



Exemplo: buscar chave bahia



Exemplo: buscar chave bahia

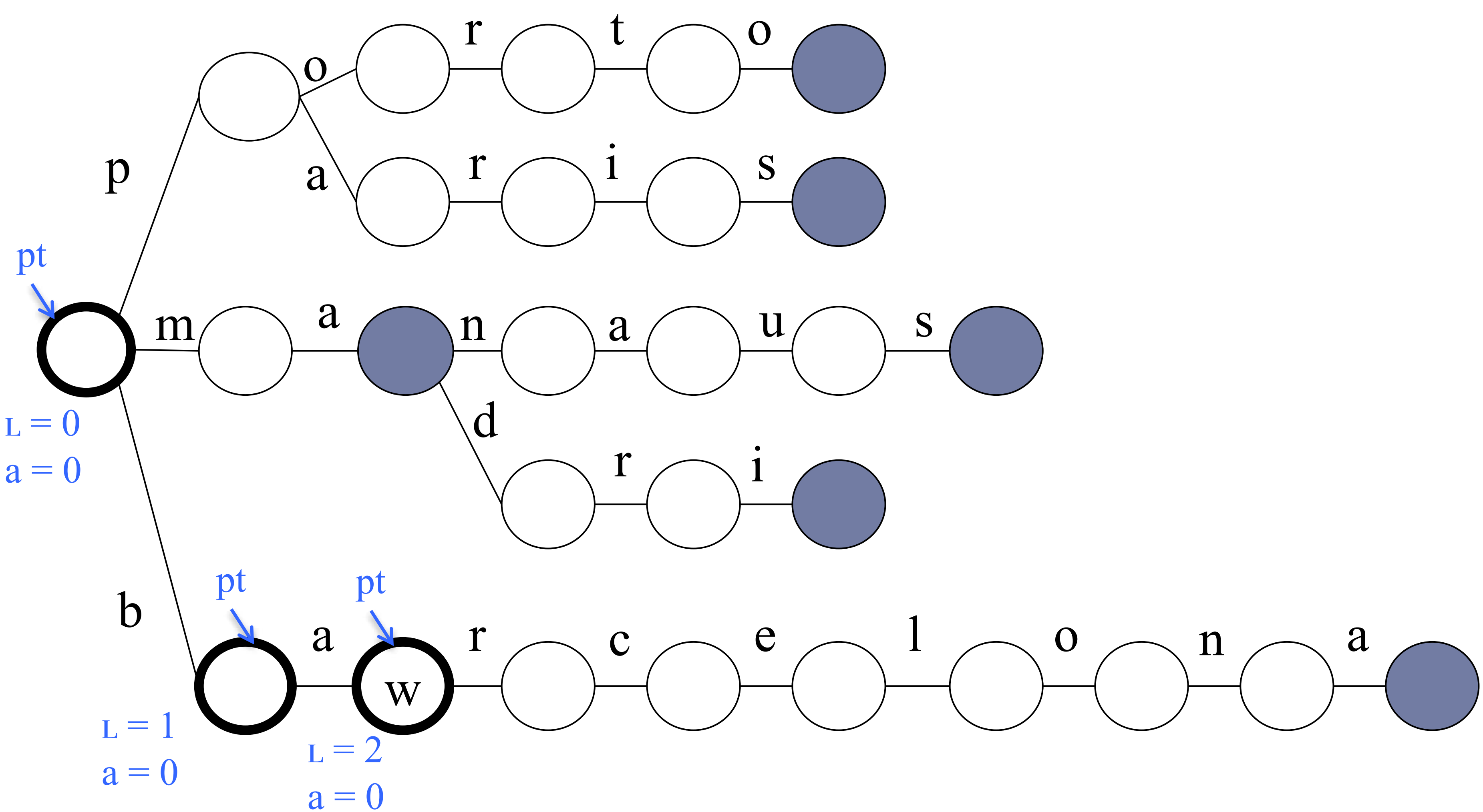


Nesse ponto não existe aresta rotulada com ”h”. Busca é encerrada, com a = 0

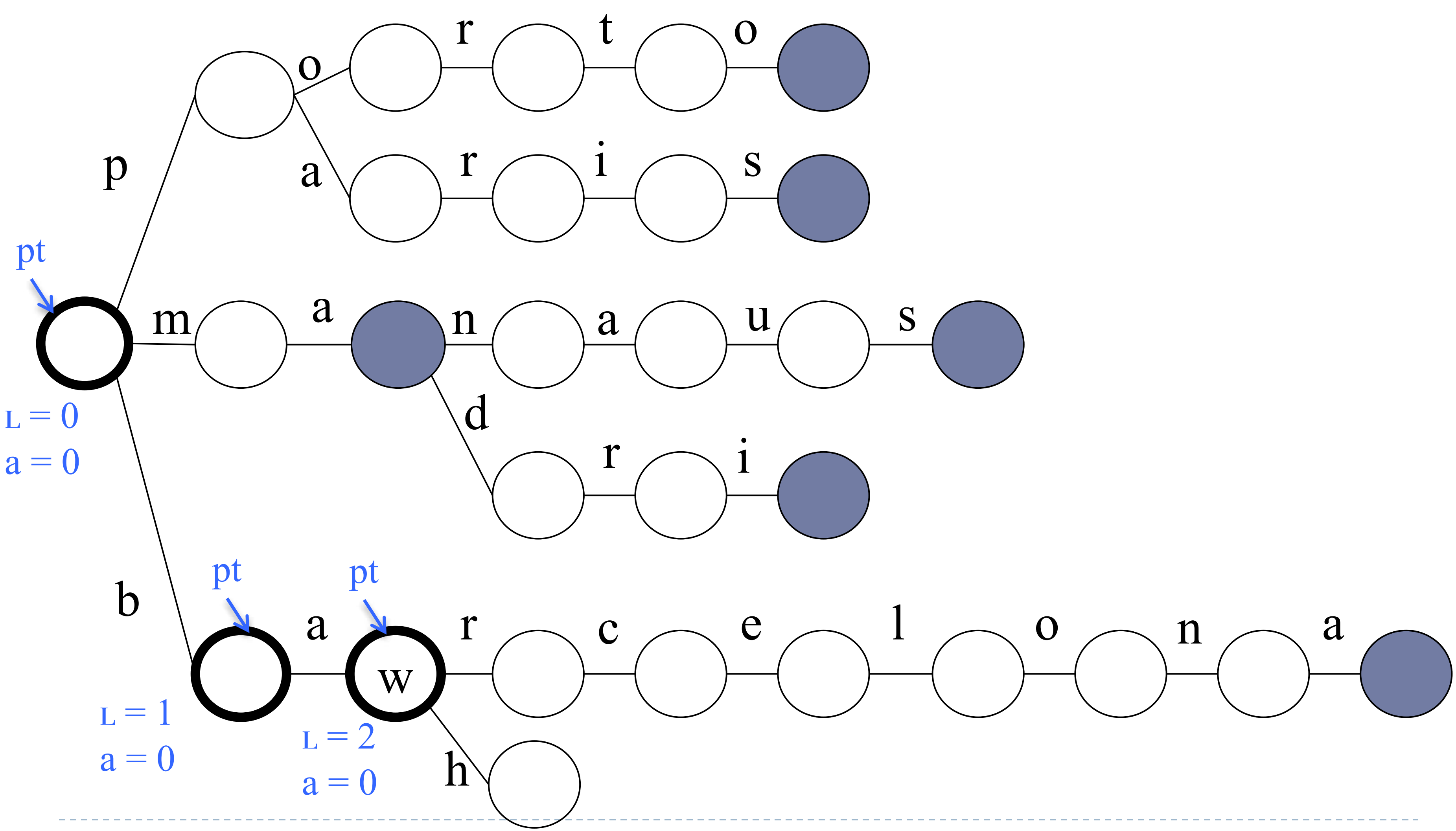
Inserção

- Para incluir uma nova chave $x = d(1), \ldots, d(k)$ numa árvore digital m-ária, utiliza-se o procedimento de busca
- Se x foi localizado, inclusão inválida
- Caso contrário, a busca determina o nó w apontado por pt, tal que o caminho da raiz até w corresponde ao maior prefixo de chave que coincide com x. O tamanho l desse prefixo também é conhecido
- Seja j a posição de $d(L+1)$ na ordenação de dígitos. Um novo nó é incluído na árvore de modo a se constituir o *j-ésimo* filho de w
- O processo se repete até que todos os dígitos de x tenham sido esgotados

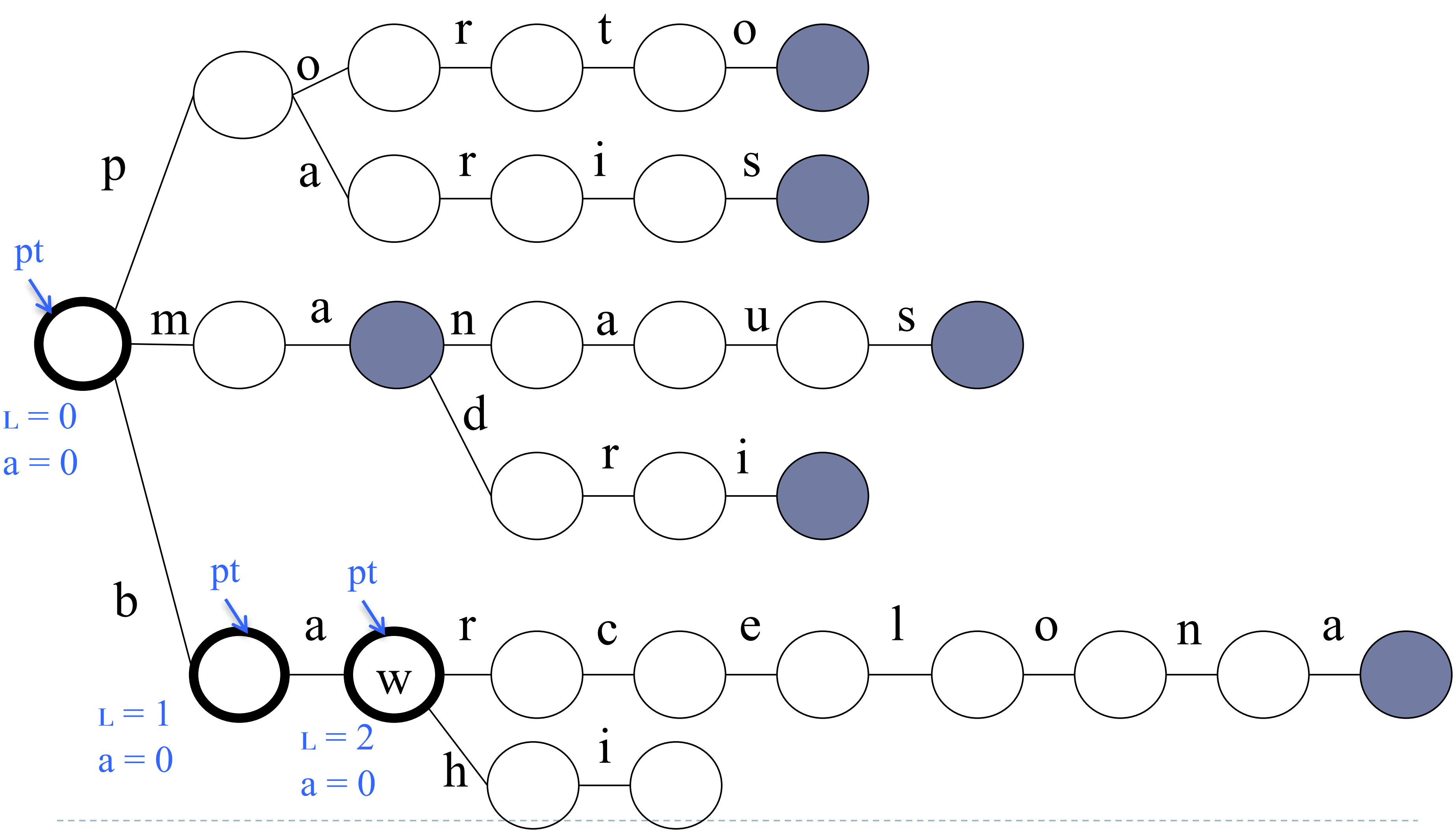
Exemplo: inserir chave bahia



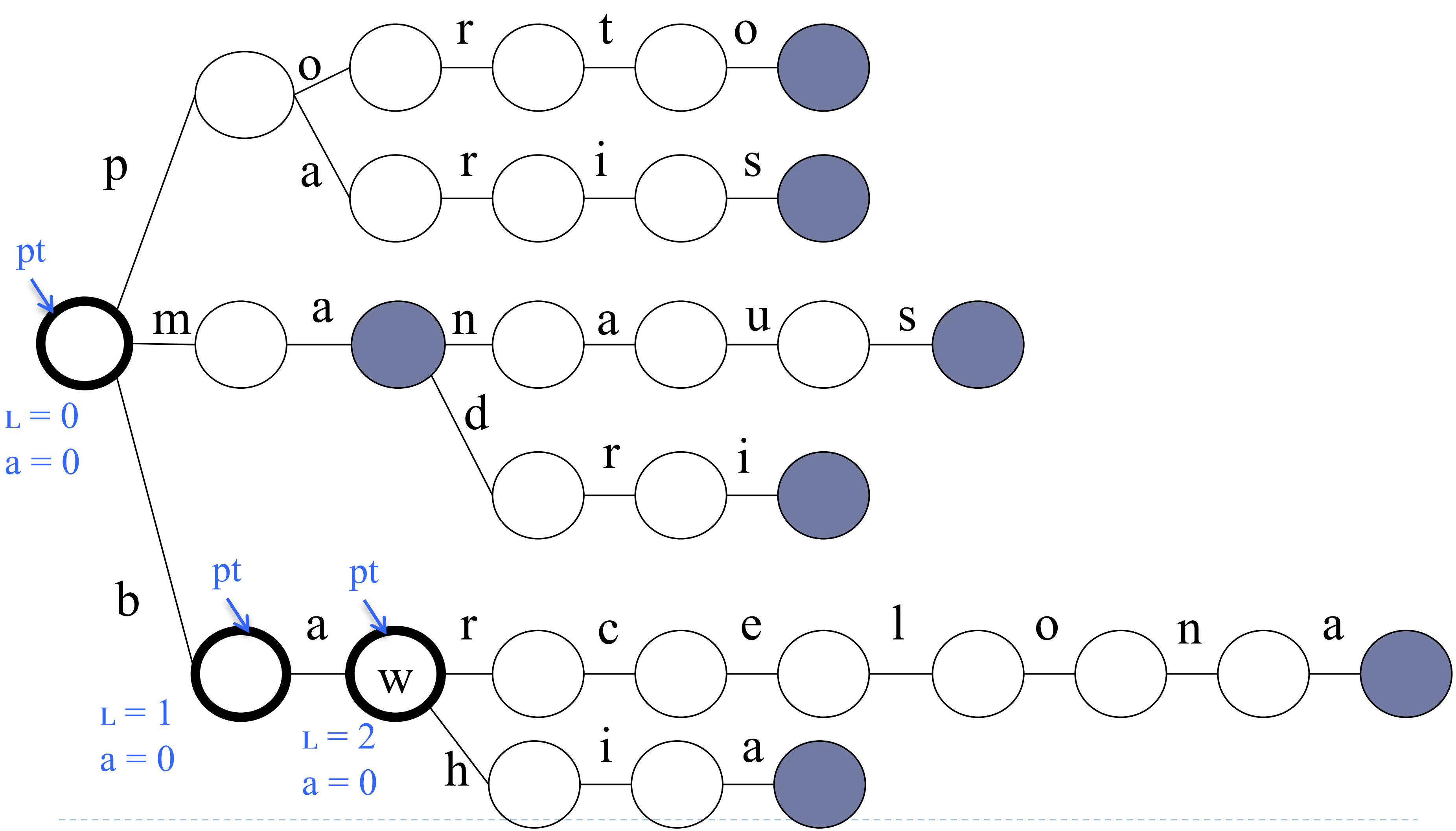
Exemplo: inserir chave bahia



Exemplo: inserir chave bahia



Exemplo: inserir chave bahia



Procedimento Insere

```
procedimento insere(x, ptraiz)
   pt := ptraiz
   L := 0
   a := 0
   buscadig(x, pt, L, a)
   se a = 0 então
      para h = L + 1, ..., k faça
         seja j a posição de d(h) na ordenação do alfabeto
         ocupar(ptz)
         para i = 1, ..., m faça ptz↑.pont[i] := λ
         pt↑.pont[j] := ptz
         ptz.info := não terminal
         pt := ptz
      pt↑.info = terminal
   senão "inclusão inválida"
```


Fonte: Algoritmo 11.2, página 262

Uso de Árvores Digitais

- Bastante utilizadas para implementar verificação ortográfica e dicionários

Considerações sobre Árvores Digitais

- Consomem muito espaço em memória
  - Para uma árvore com **n** chaves de tamanho máximo **t** com prefixos distintos, o espaço necessário seria **O(n m t)**
- Árvores com muitos zigue-zagues quase sempre são ineficientes (um zigue-zague de uma árvore *T* é uma subárvore parcial cujos nós possuem apenas um único filho em *T*)

Busca Digital

- Árvore Digital
- **Árvore Digital Binária**
- Árvore Patrícia (implementação eficiente de Árvore Digital Binária)

Árvore Digital Binária

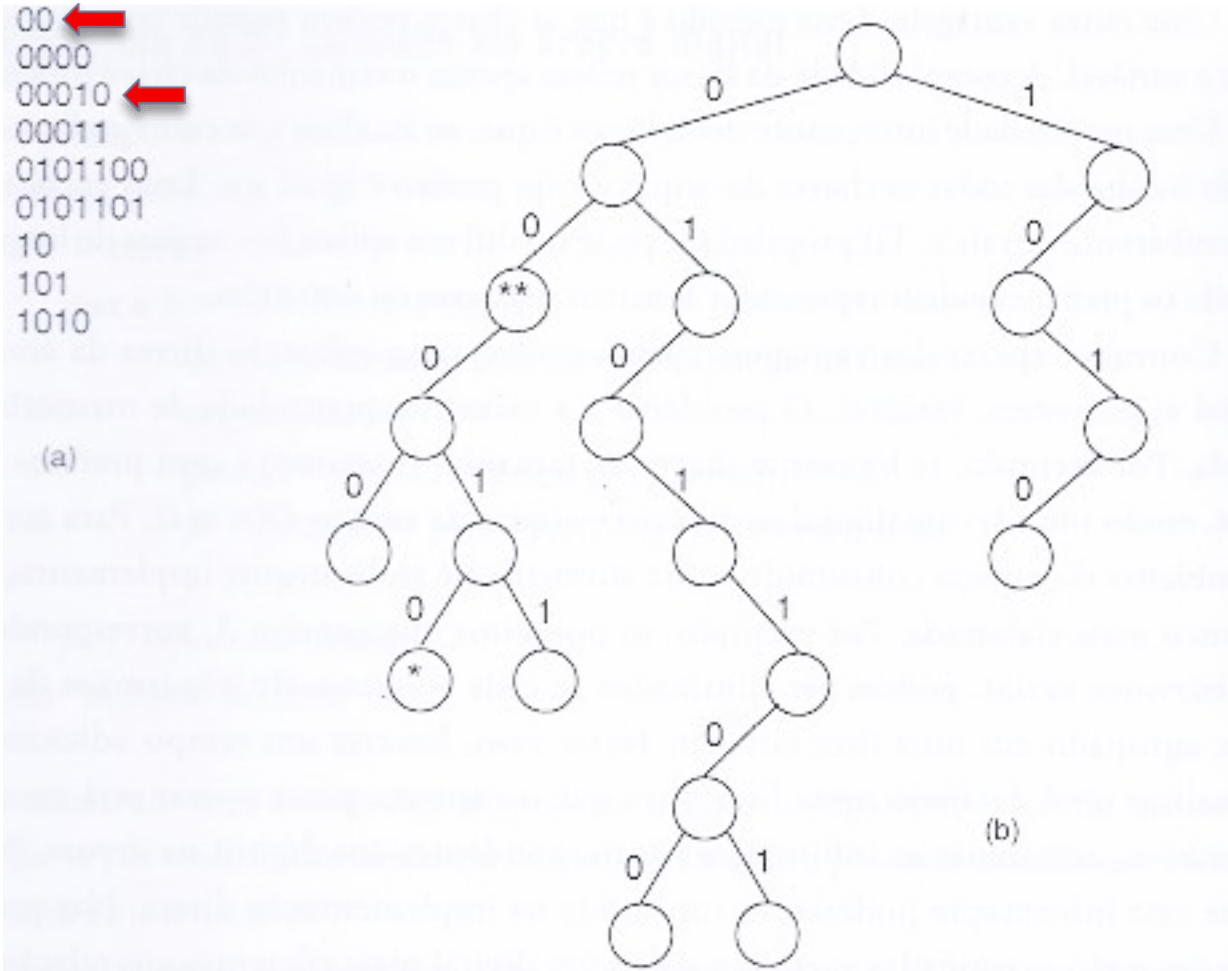
- É simplesmente o caso binário de uma árvore digital, onde m = 2
- Alfabeto é representado por {0,1}

- Vantagens:
  - Número de ponteiros NULL é menor (ocupa menos espaço)
  - Chaves podem ser facilmente convertidas em binário
- Desvantagem:
  - Zigue-zagues ainda podem ocorrer

Árvore Binária de Prefixos

- É uma Árvore Digital Binária onde nenhuma chave é prefixo de outra
- Propriedade interessante: todas as chaves estão nas folhas

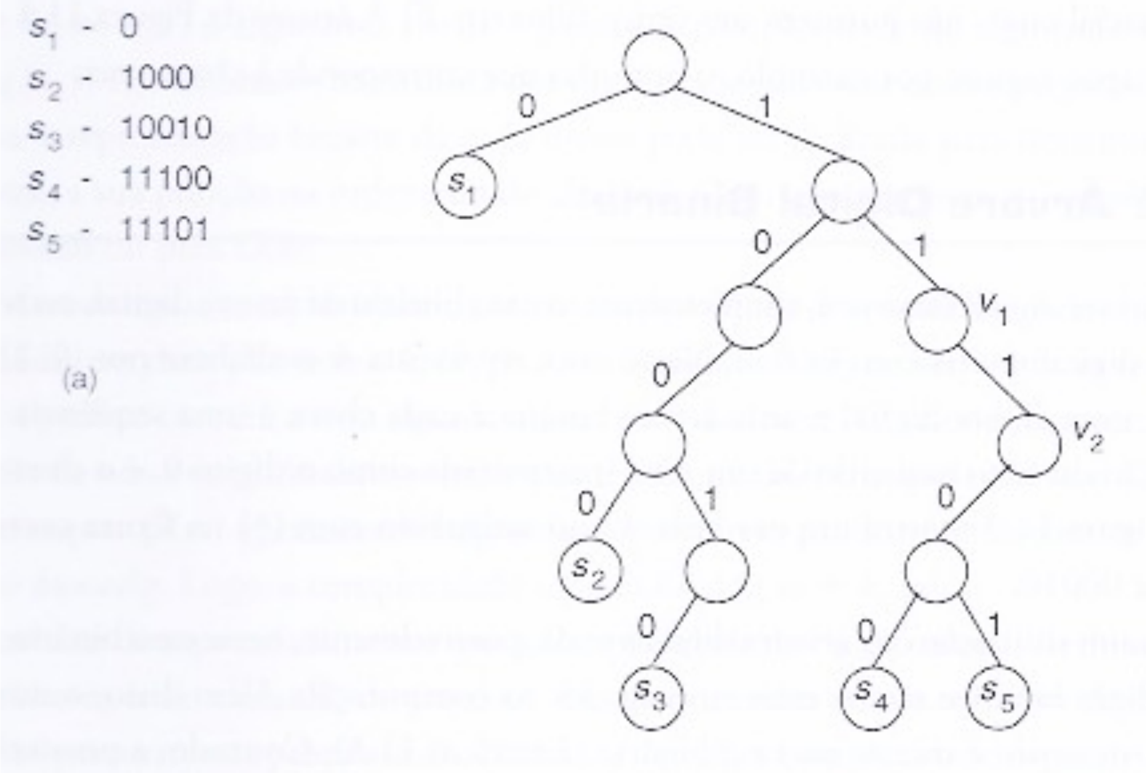
Contra Exemplo



Não é Árvore de Prefixo, pois o nó ** é prefixo do nó *

Fonte: Figura 11.2, pág. 254

Exemplo



É Árvore de Prefixo, pois nenhuma chave é prefixo de outra. Todas as chaves estão nas folhas.

Uso de árvores de Prefixo

- São muito usadas para aplicações de codificação e compressão
- Nas próximas aulas aprenderemos a usar esse tipo de árvore para gerar códigos únicos onde um não é prefixo do outro (propriedade necessária para viabilizar a decodificação)

Busca Digital

- Árvore Digital
- Árvore Digital Binária
- **Árvore Patrícia** (implementação eficiente de Árvore Digital Binária)

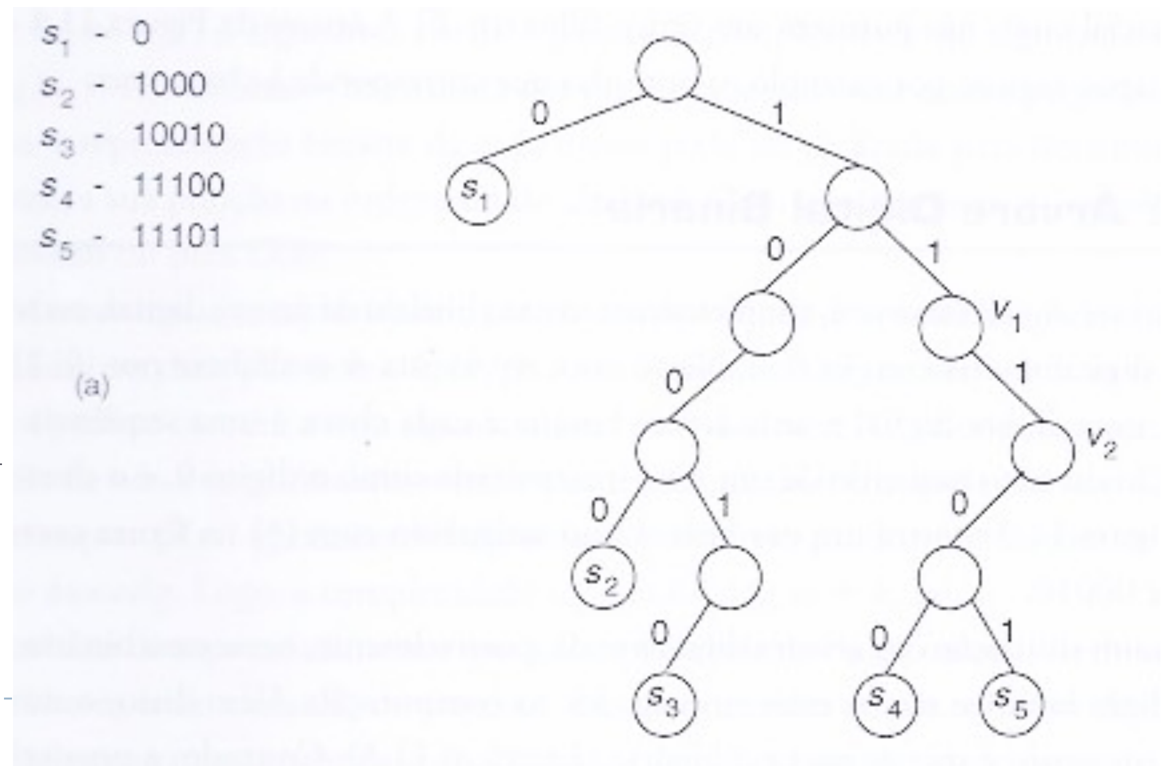
Árvore Patrícia

- Construída a partir da árvore binária de prefixos

- $S = \{s_1, \ldots, s_n\}$ são as chaves indexadas pela árvore
- Nenhuma chave s_i é prefixo de outra

Motivação: Busca por s_4 (11100) em Árvore Digital Binária

- A certo ponto do caminho, a busca entra em um zigue-zague (v_1, v_2)
- Em cada nó do zigue-zague, há sempre uma única opção de caminho
- Ideia:
  - Ao chegar em v_1 pular para o quinto dígito
  - É preciso adicionar essa informação em v_1



Árvore Patrícia

- Cada vértice v possui um rótulo r(v) que será usado para “cortar caminho”

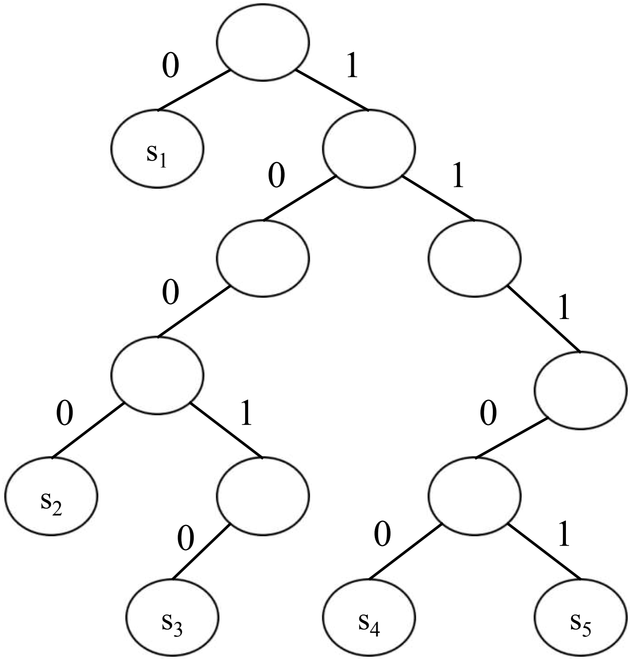
Construção de Árvore Patrícia

- Seja H a Árvore de Prefixos original
- Construir a Árvore Patrícia T da seguinte forma:
  - Para cada zigue-zague $v_1, \ldots, v_k$ em H:
    - Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.
    - Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$
  - Se algum vértice v de T permaneceu sem rótulo, definir rótulo $r(v) = nível_H(v)$

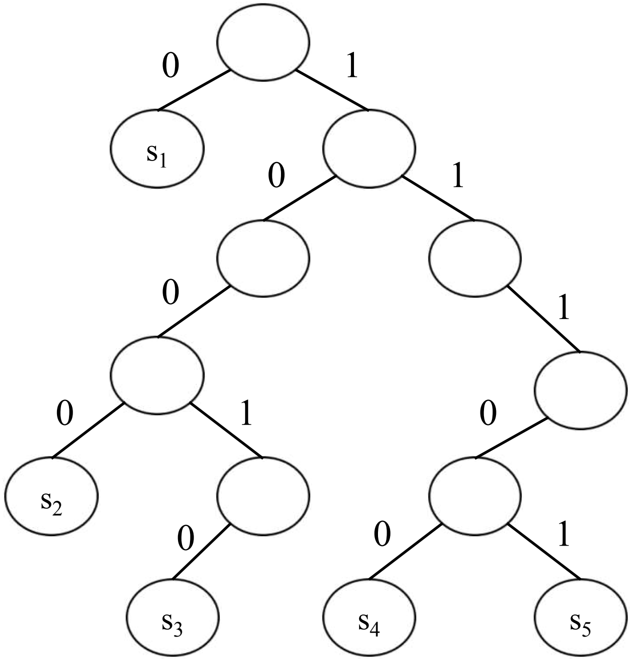
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvores H e T

Árvore de Prefixos H



Árvore Patrícia T correspondente

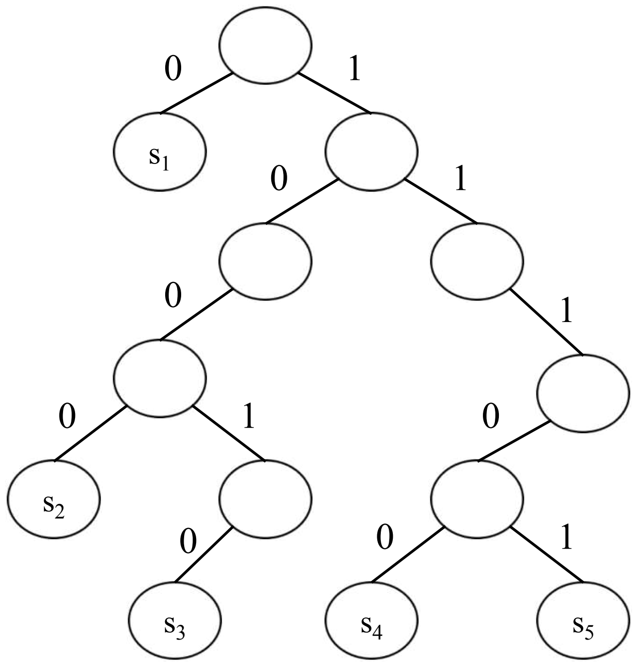


Para cada zigue-zague $v_1, \ldots, v_k$ em H:
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.
Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

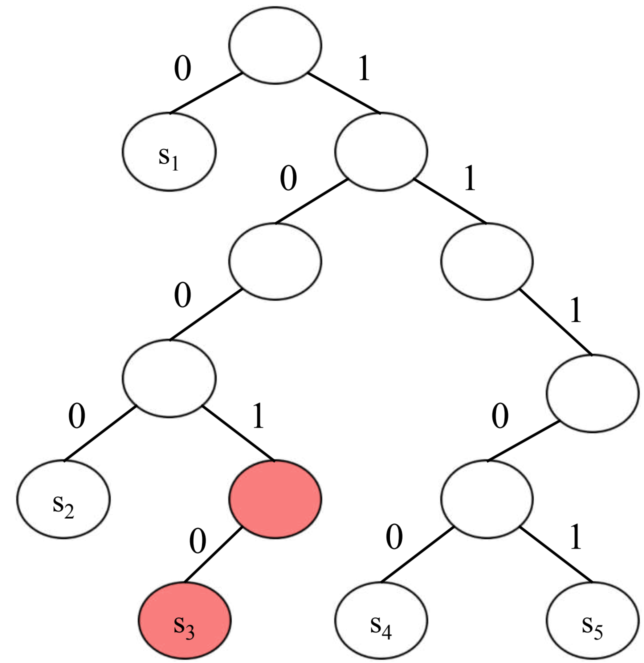
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvores H e T

Árvore de Prefixos H



Árvore Patrícia T correspondente



Para cada zigue-zague $v_1, \ldots, v_k$ em H:

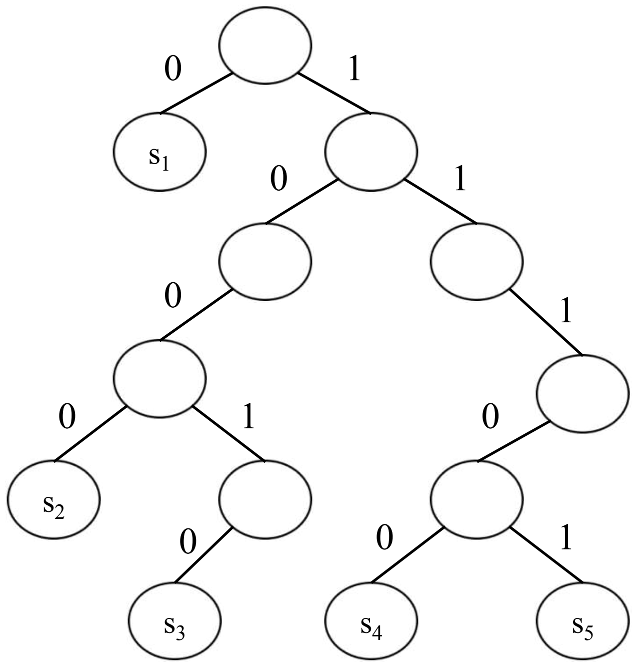
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.

Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = \text{nivel}_H(v_1) + k$

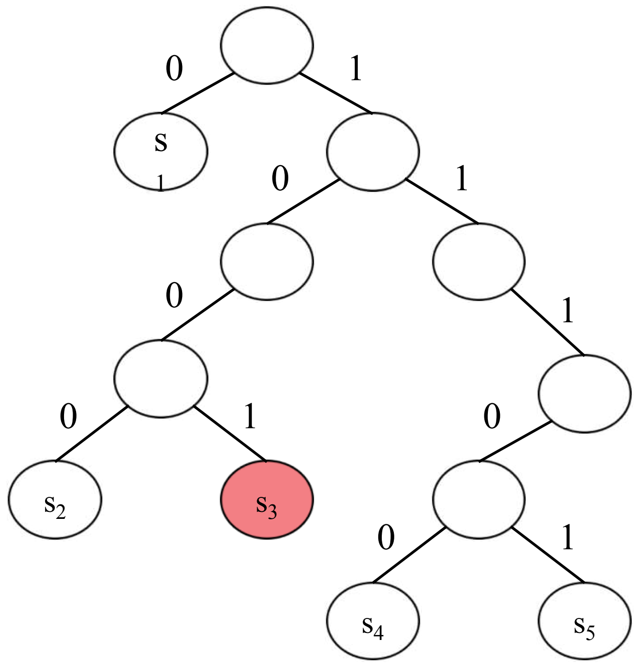
Árvores H e T

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvore de Prefixos H



Árvore Patrícia T correspondente



Para cada zigue-zague $v_1, \ldots, v_k$ em H:

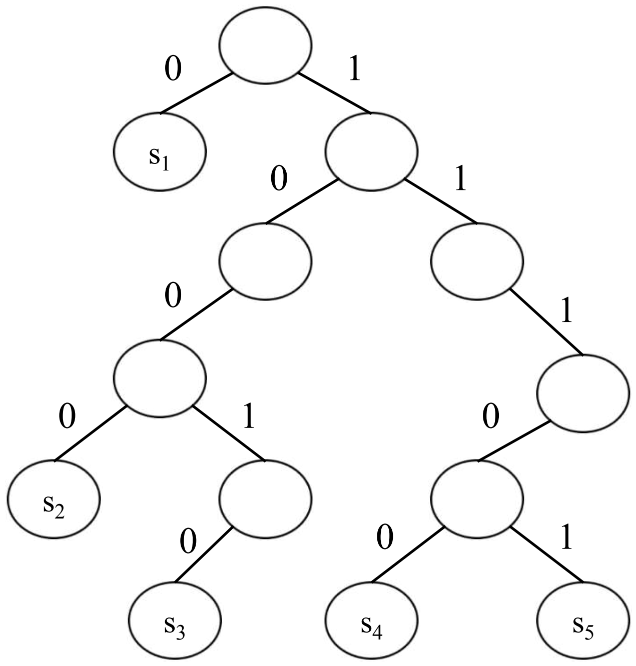
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.

Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

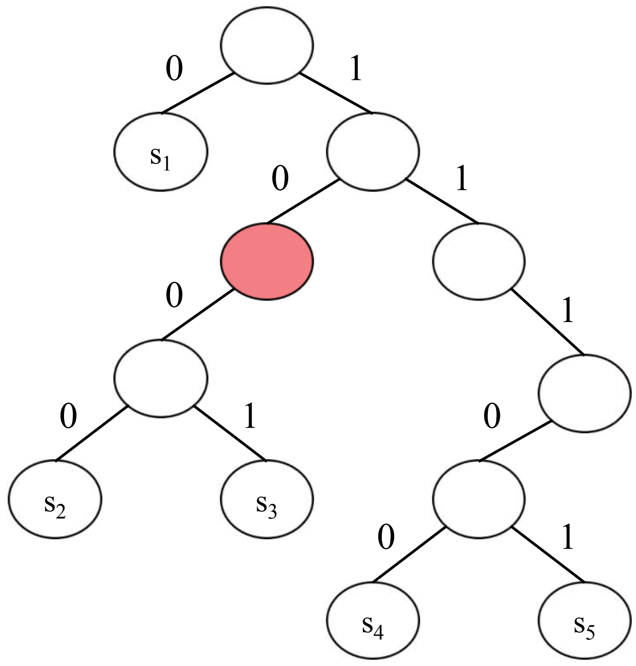
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvores H e T

Árvore de Prefixos H



Árvore Patrícia T correspondente



Para cada zigue-zague $v_1, \ldots, v_k$ em H:

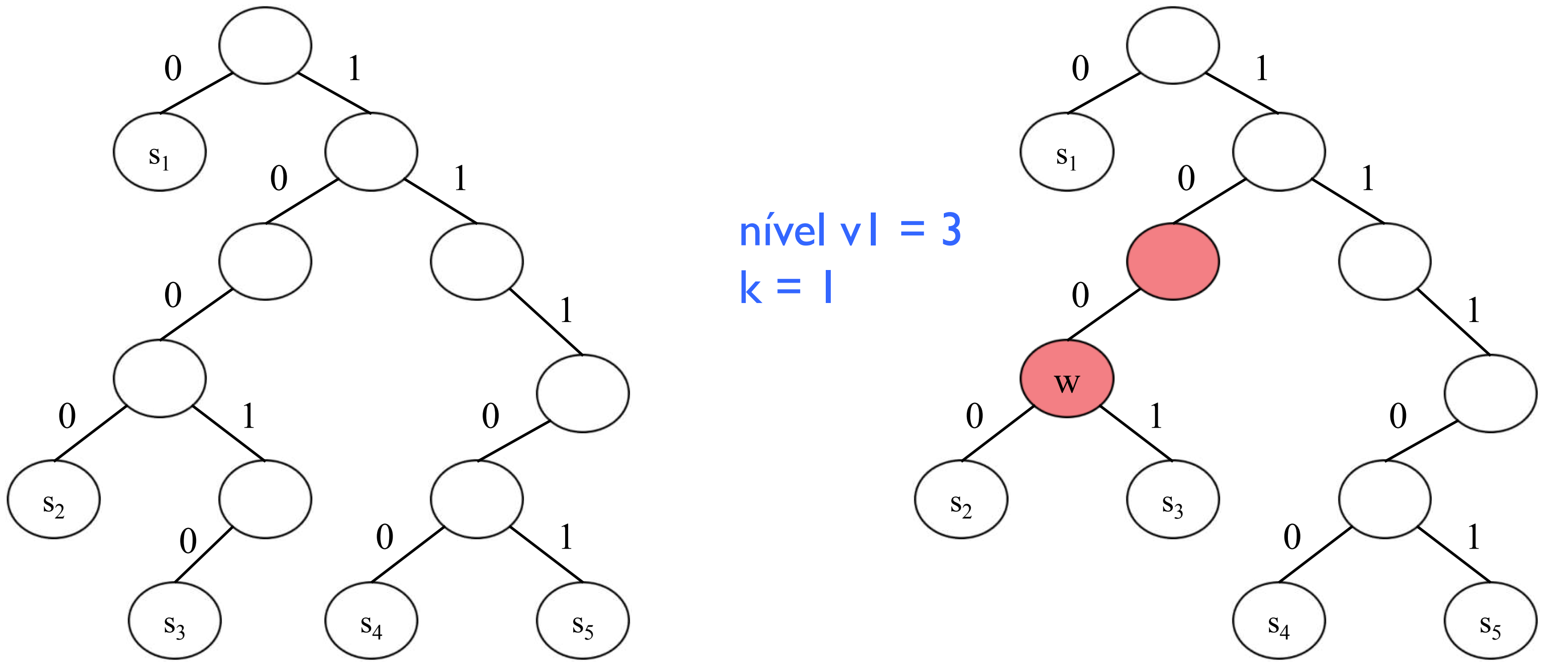
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.

Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvores H e T

Árvore de Prefixos H   Árvore Patrícia T correspondente



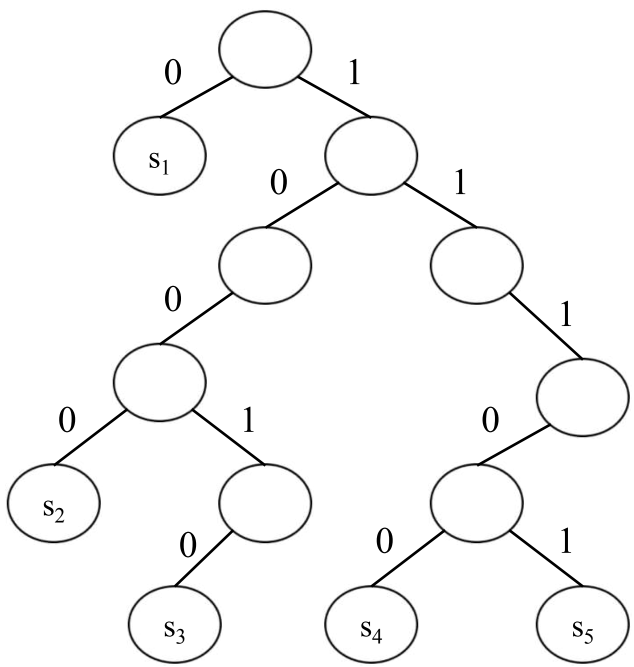
nível v1 = 3
k = 1

Para cada zigue-zague $v_1, \ldots, v_k$ em H:
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.
Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

Árvores H e T

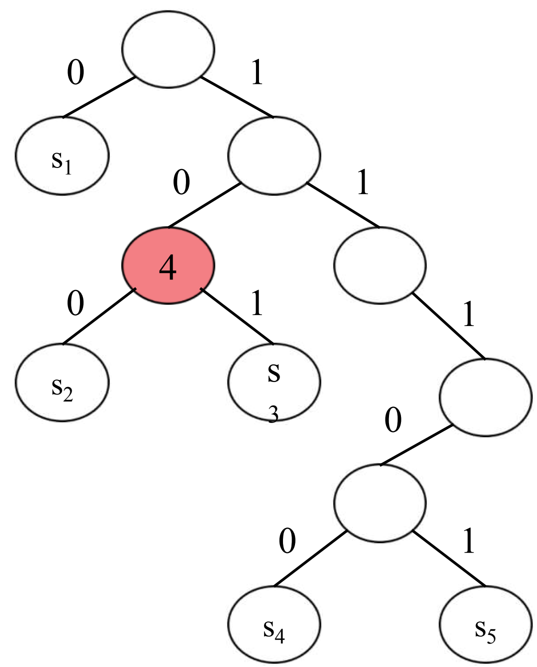
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvore de Prefixos H



Árvore Patrícia T correspondente

nível v1 = 3
k = 1

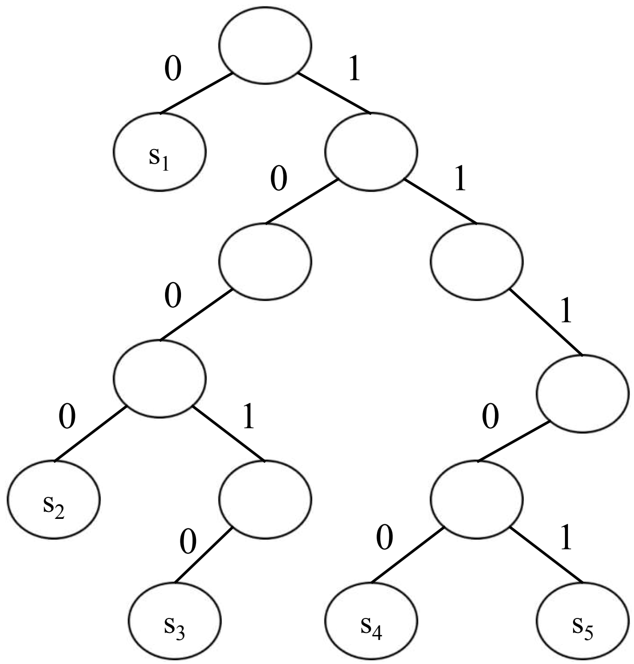


Para cada zigue-zague $v_1, \ldots, v_k$ em H:
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.
Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

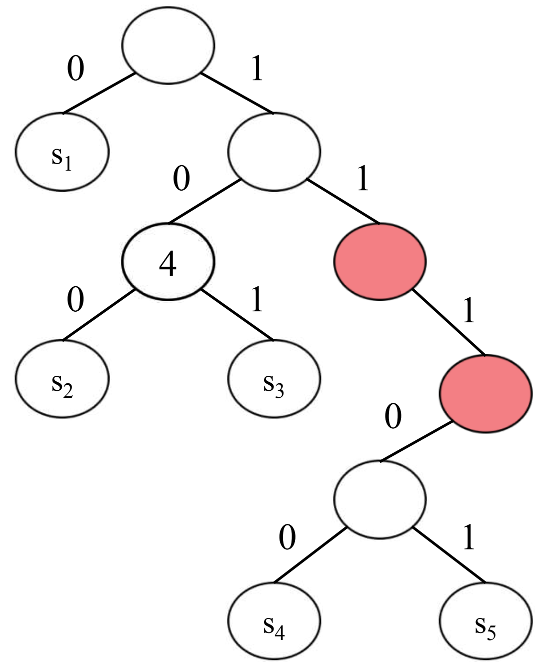
Árvores H e T

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvore de Prefixos H



Árvore Patrícia T correspondente



Para cada zigue-zague $v_1, \ldots, v_k$ em H:

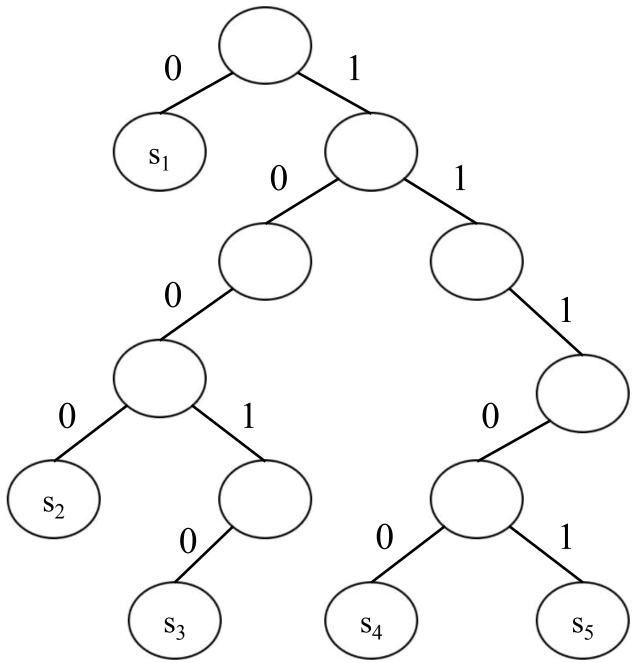
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.

Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

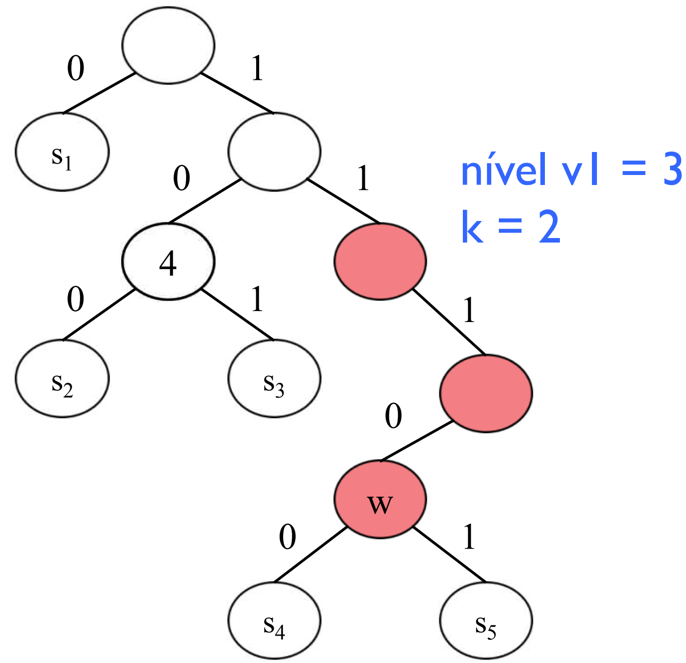
Árvores H e T

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvore de Prefixos H



Árvore Patrícia T correspondente



Para cada zigue-zague $v_1, \ldots, v_k$ em H:

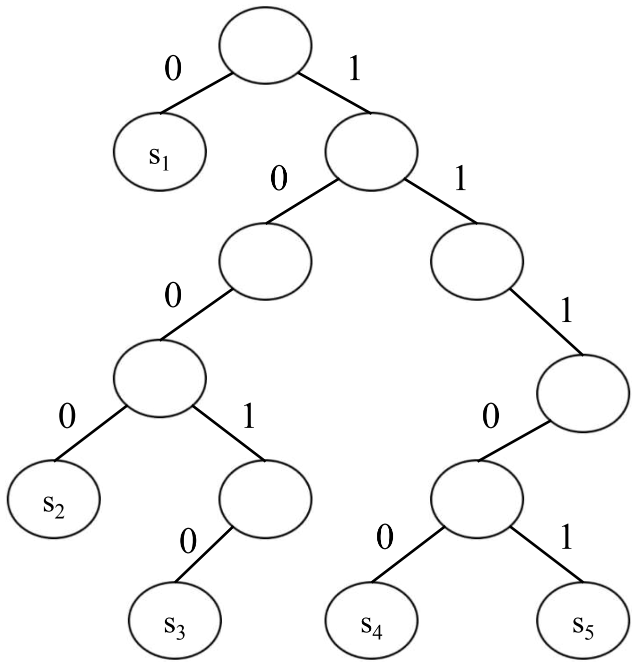
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.

Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

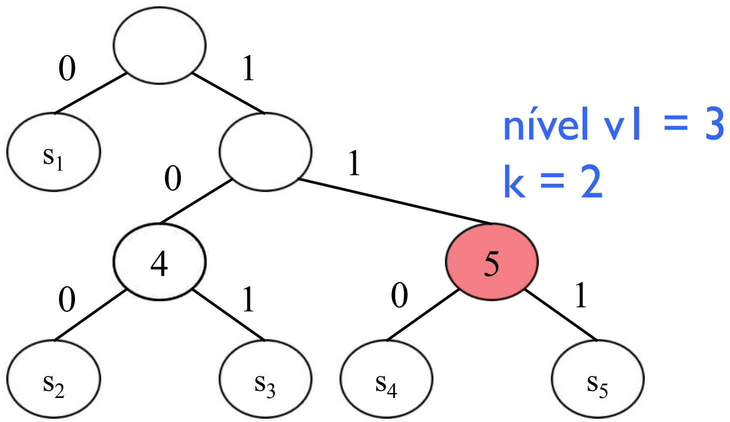
Árvores H e T

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvore de Prefixos H



Árvore Patrícia T correspondente

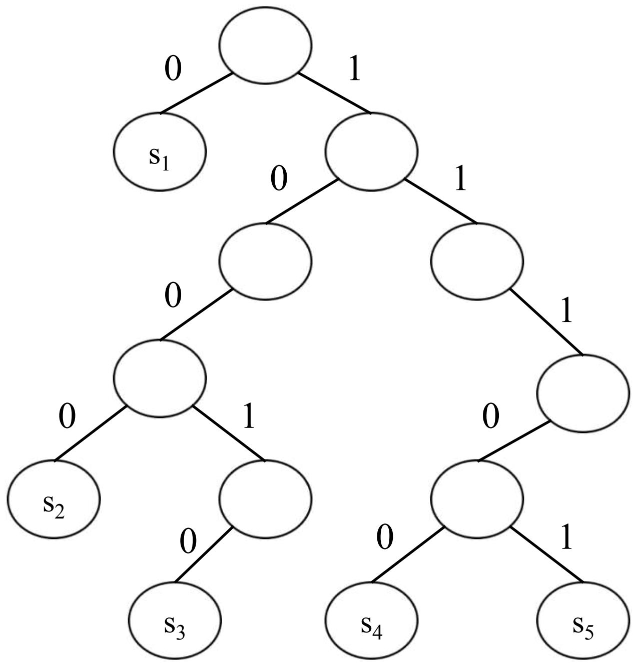


Para cada zigue-zague $v_1, \ldots, v_k$ em H:
Se v_k é um nó folha referente à chave s_i, compactar $v_1, \ldots, v_k$ em v_1 e definir $r(v_1) = s_i$.
Caso contrário, v_k possui o filho w. Compactar $v_1, \ldots, v_k$, w em v_1 e definir $r(v_1) = \text{nivel}_H(v_1) + k$

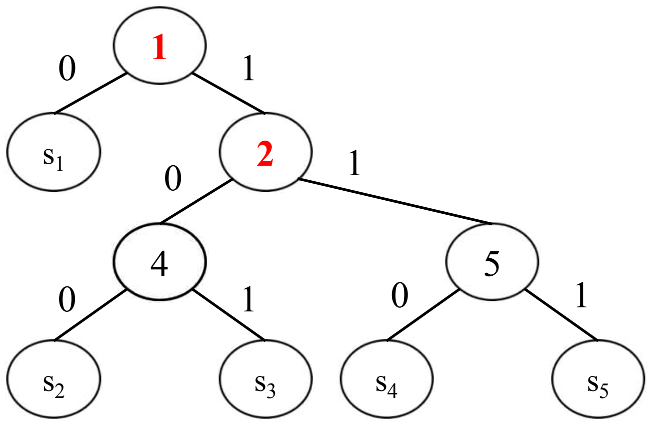
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

Árvores H e T

Árvore de Prefixos H



Árvore Patrícia T correspondente

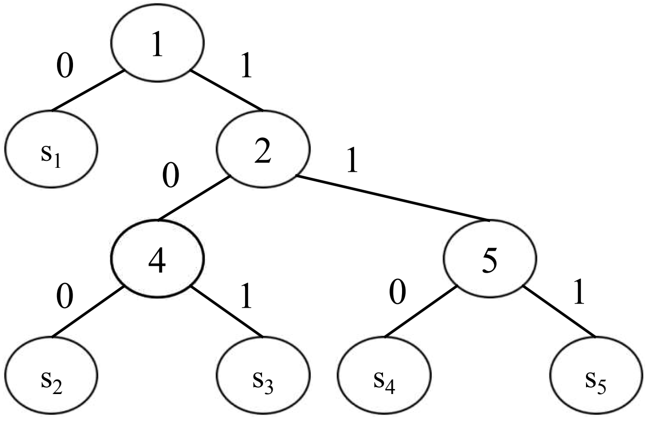


Se algum vértice v de T permaneceu sem rótulo, definir rótulo $r(v) = \text{nível}_H(v)$

Propriedade dos Rótulos da Árvore Patrícia

- Em uma busca por chave x, o rótulo de um nó v, não folha, é o índice do dígito de x relativo a v

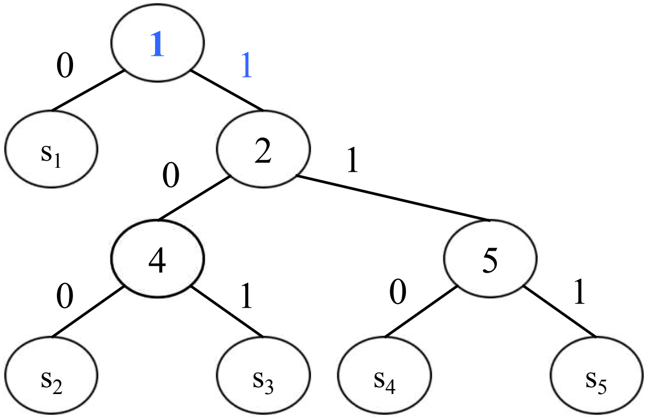
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Exemplo: buscar chave 11101

- 11101

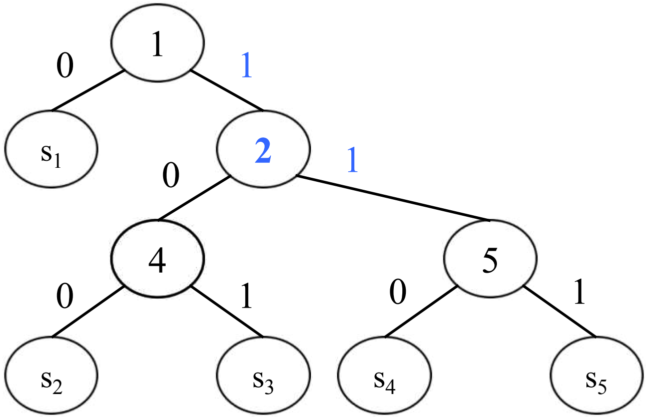
s_1 = 0
s_2 = 1000
s_3 = 10010
s_4 = 11100
s_5 = 11101



Exemplo: buscar chave 11101

- 11101

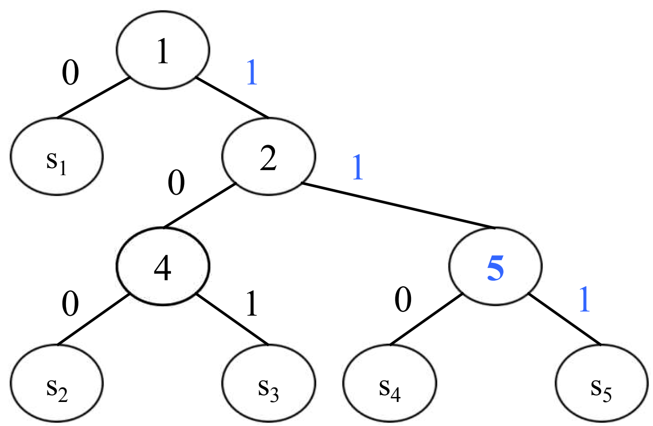
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Exemplo: buscar chave 11101

- 11101

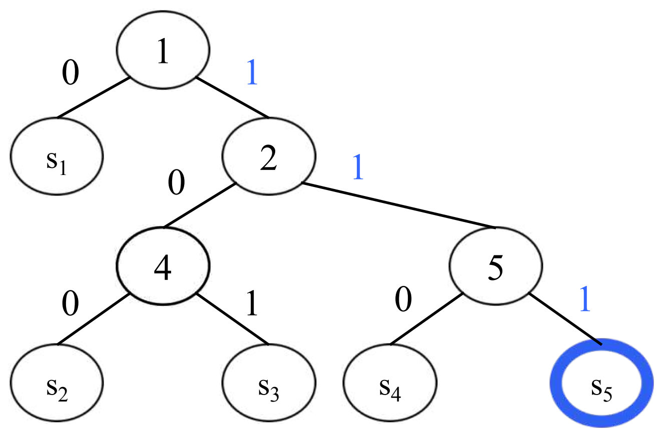
s_1 = 0
s_2 = 1000
s_3 = 10010
s_4 = 11100
s_5 = 11101



Exemplo: buscar chave 11101

- 11101

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Procedimento Busca

```
/* procedimento retorna a = 1 se chegar a uma folha
    nesse caso ainda é necessário comparar x com o rótulo
    do nó apontado por pt para concluir que a chave x foi
    encontrada
*/
procedimento buscapat(x, pt, a)
   se pt↑.esq = λ
   então a := 1
   senão se k < pt↑.r então a := 2
             senão se d[pt↑.r] = 0 então
                         pt := pt↑.esq
                         buscapat(x, pt, a)
                      senão pt := pt↑.dir
                            buscapat(x, pt, a)
```


Fonte: Algoritmo 11.3, página 267

Inserção numa Árvore Patrícia T

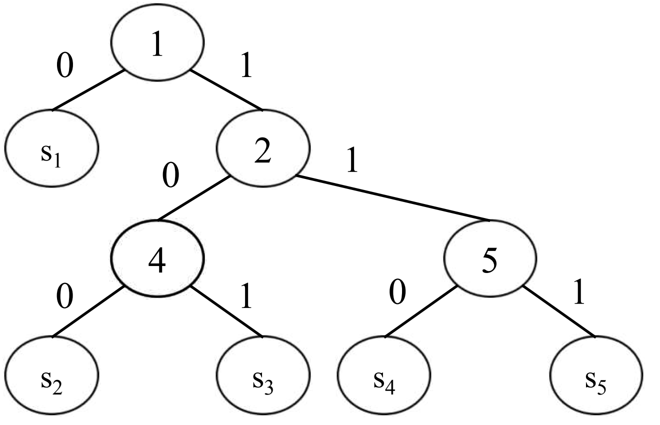
- Efetuar busca da chave **x** em **T** (**x** tem comprimento **k**)
- A busca termina num nó **y** (interno ou folha) de **T**
- Selecionar **y'**, um dos nós folha descendentes de **y**. Seja s_i a chave contida em **y'** (Se **y** é folha, **y' = y**)
- Seja L o comprimento do maior prefixo comum de **x** e s_i (ou seja, **x** e s_i coincidem exatamente até o **L-ésimo** dígito)
- Seja **c** o comprimento de s_i
- Se L **= k**, inserção inválida (**x** é prefixo de s_i)
- Se L **= c**, inserção inválida (s_i é prefixo de **x**)

Se inserção for válida

- Determinar nó **z** de **T** onde será feita a inclusão
- Se **y'** é o único nó em **T**, então **z = y'**
- Senão, seja **z'** o pai de **y'**, e **A** o caminho da raiz de **T** até **z'**
- Se $\mathbf{r(z')} \leq L + \mathbf{1}$ então **z = y'**
- Quando $\mathbf{r(z')} > L + \mathbf{1}$, **z** será o nó de **A** mais próximo da raiz de **T**, tal que $\mathbf{r(z)} > L + \mathbf{1}$
- Criar dois nós novos, **v** e **w**, com rótulos $\mathbf{r(v)} = L + \mathbf{1}$, $\mathbf{r(w)} = \mathbf{x}$. O pai de **v** será o antigo pai de **z**. Os filhos de **v** serão **w** e **z**, sendo **w** o filho esquerdo se $\mathbf{d}(L + \mathbf{1}) = \mathbf{0}$ ou o direito quando $\mathbf{d}(L + \mathbf{1}) = \mathbf{1}$
- Se **z** era a raiz de **T**, a nova raiz passa a ser **v**.

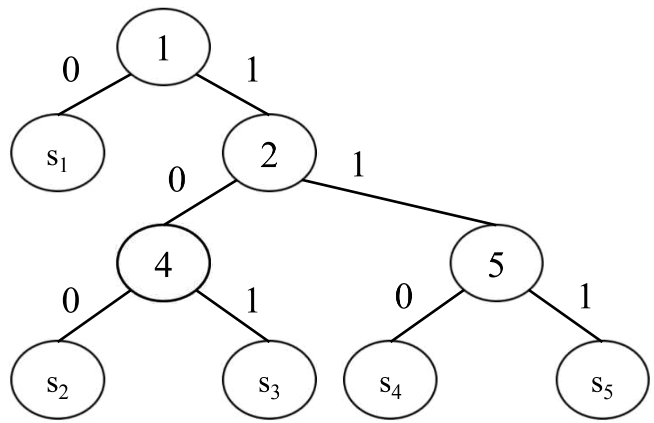
Exemplo: inserção inválida
Inserir chave 10

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Exemplo: inserção inválida Inserir chave 10

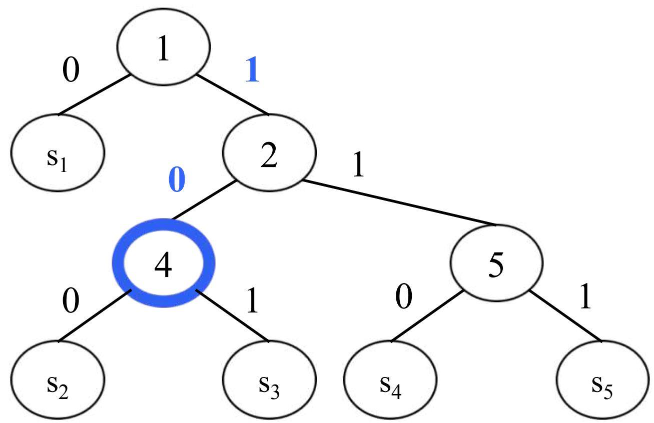
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Buscar 10

Exemplo: inserção inválida
Inserir chave 10

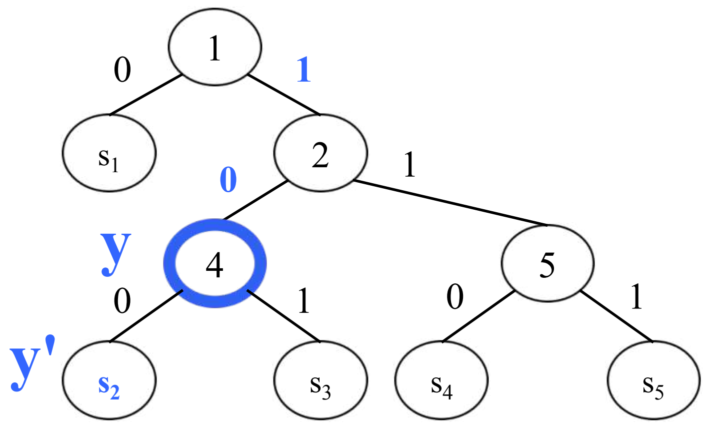
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Buscar 10

Exemplo: inserção inválida
Inserir chave 10

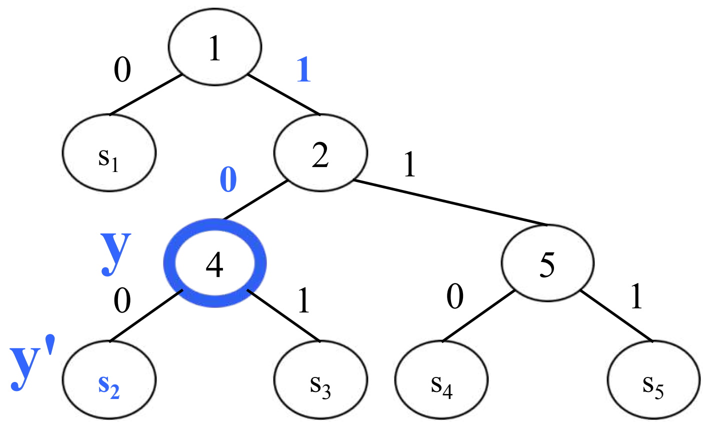
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Buscar 10
A busca termina num nó y (interno ou folha) de T
Selecionar y', um dos nós folha descendentes de y.
Seja s_i a chave contida em y' (Se y é folha, y' = y)

Exemplo: inserção inválida
Inserir chave 10

s_1 = 0
s_2 = 1000
s_3 = 10010
s_4 = 11100
s_5 = 11101

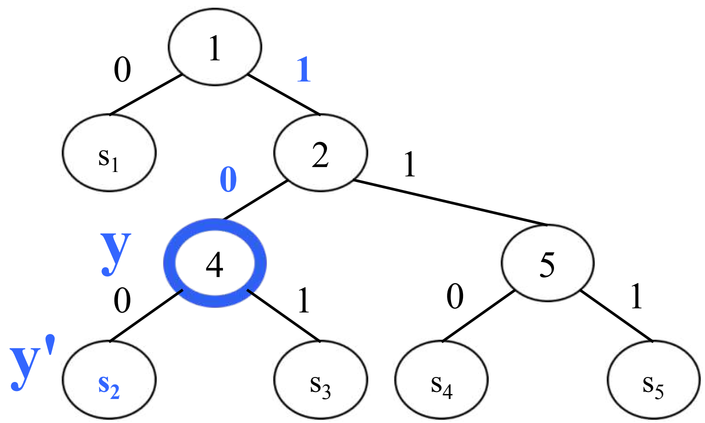


L = **2**
k = 2
c = 4

Seja L o comprimento do maior prefixo comum de **x** e $\mathbf{s_i}$ (ou seja, **x** e $\mathbf{s_i}$ coincidem exatamente até o **L-ésimo** dígito)
Seja **c** o comprimento de $\mathbf{s_i}$
Se L = **k**, inserção inválida (**x** é prefixo de $\mathbf{s_i}$)
Se L = **c**, inserção inválida ($\mathbf{s_i}$ é prefixo de **x**)

Exemplo: inserção inválida
Inserir chave 10

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



$L = 2$
$k = 2$
$c = 4$

Seja L o comprimento do maior prefixo comum de $\mathbf{x}$ e $\mathbf{s_i}$ (ou seja, $\mathbf{x}$ e $\mathbf{s_i}$ coincidem exatamente até o **L-ésimo** dígito)

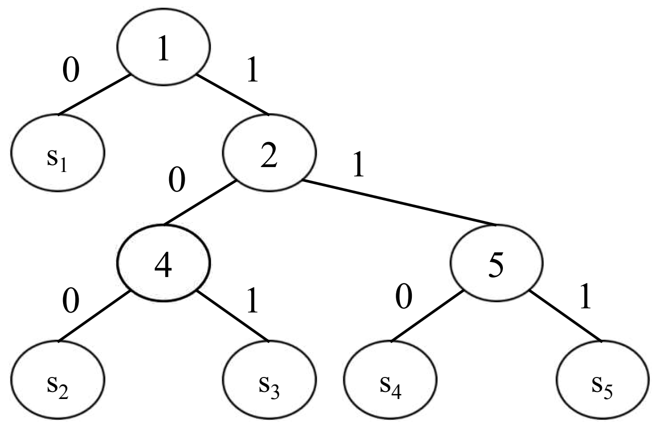
Seja $\mathbf{c}$ o comprimento de $\mathbf{s_i}$

Se $L = \mathbf{k}$, inserção inválida ($\mathbf{x}$ é prefixo de $\mathbf{s_i}$)

Se $L = \mathbf{c}$, inserção inválida ($\mathbf{s_i}$ é prefixo de $\mathbf{x}$)

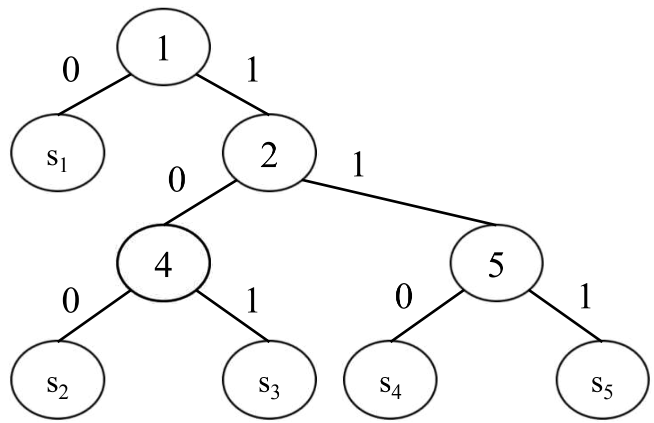
Exemplo: inserção válida
Inserir chave 111101

s_1 = 0
s_2 = 1000
s_3 = 10010
s_4 = 11100
s_5 = 11101



Exemplo: inserção válida
Inserir chave 111101

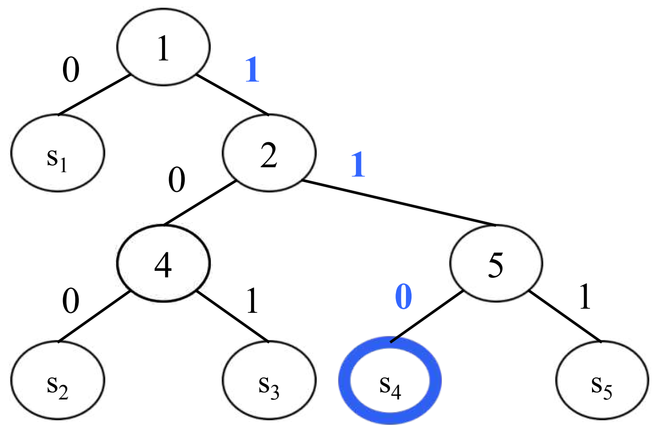
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Buscar 111101

Exemplo: inserção válida
Inserir chave 111101

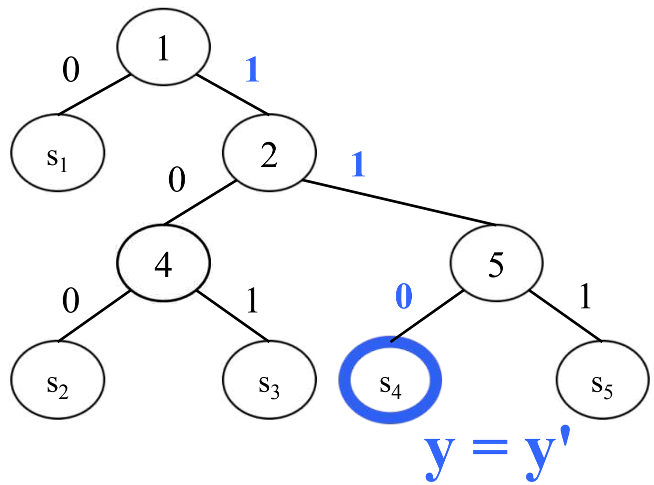
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Buscar **11**1**10**1

Exemplo: inserção válida
Inserir chave 111101

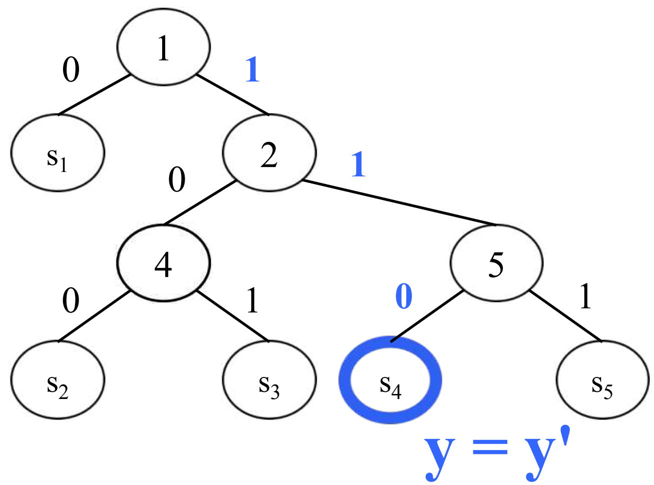
$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$



Buscar 111101
A busca termina num nó y (interno ou folha) de T
Selecionar y’, um dos nós folha descendentes de y.
Seja s_i a chave contida em y’ (Se y é folha, y’ = y)

Exemplo: inserção válida
Inserir chave 111101

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$

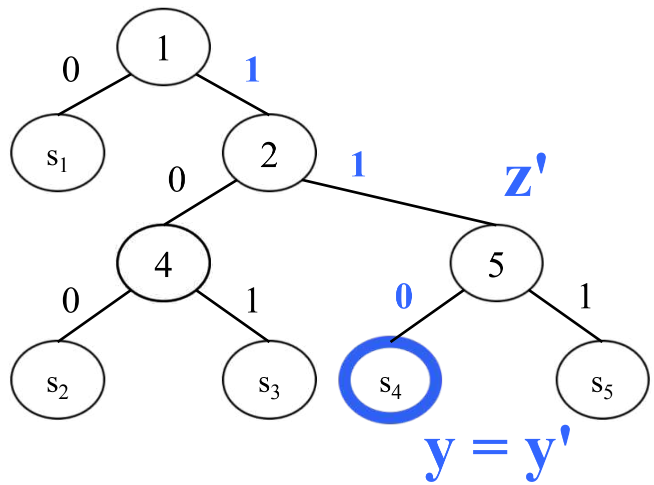


$\text{L} = 3$
$k = 6$
$c = 5$

Seja **L** o comprimento do maior prefixo comum de **x** e $\mathbf{s_i}$ (ou seja, **x** e $\mathbf{s_i}$ coincidem exatamente até o **L-ésimo** dígito)
Seja **c** o comprimento de $\mathbf{s_i}$
Se **L** = **k**, inserção inválida (**x** é prefixo de $\mathbf{s_i}$)
Se **L** = **c**, inserção inválida ($\mathbf{s_i}$ é prefixo de **x**)

Exemplo: inserção válida
Inserir chave 111101

s_1 = 0
s_2 = 1000
s_3 = 10010
s_4 = 11100
s_5 = 11101

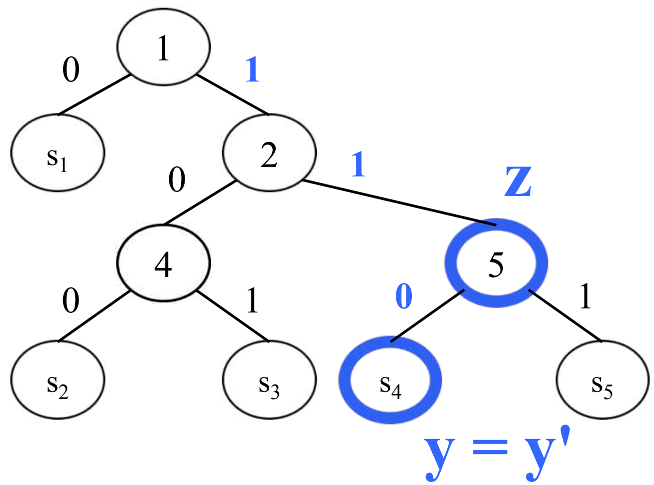


$L = 3$
$k = 6$
$c = 5$

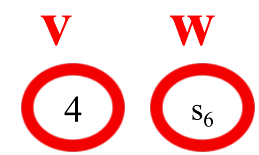
Determinar nó **z** de **T** onde será feita a inclusão
Se **y'** é o único nó em **T**, então **z = y'**
Senão, seja **z'** o pai de **y'**, e **A** o caminho da raiz de **T** até **z'**
Se **r(z')** $\leq$ L + 1 então **z = y'**
Quando **r(z')** > L + 1, **z** será o nó de **A** mais próximo da raiz de **T**, tal que **r(z)** > L + 1

Exemplo: inserção válida
Inserir chave 111101

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$
$s_6 = 111101$



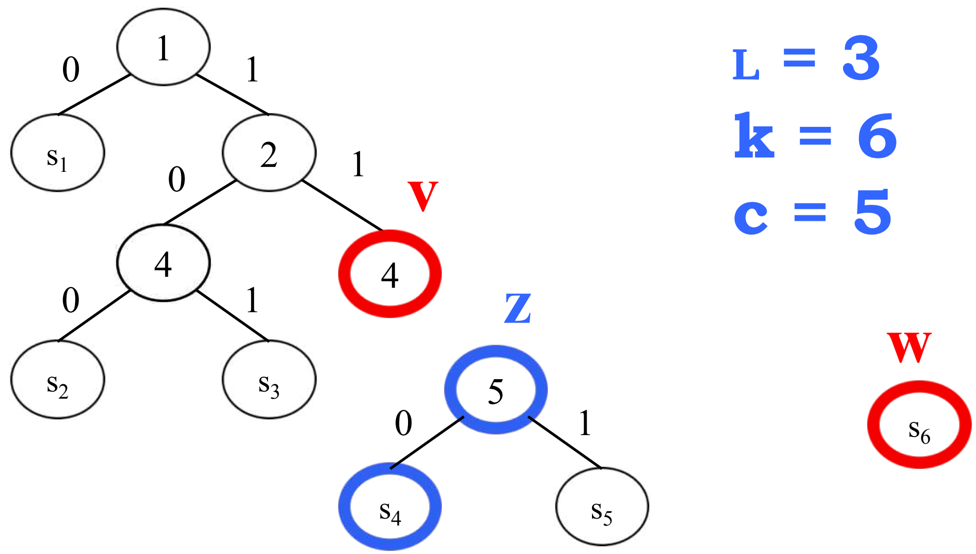
$\text{L} = 3$
$k = 6$
$c = 5$



Criar dois nós novos, **v** e **w**, com rótulos **r(v) = L + 1**, **r(w) = x**.

Exemplo: inserção válida
Inserir chave 111101

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$
$s_6 = 111101$

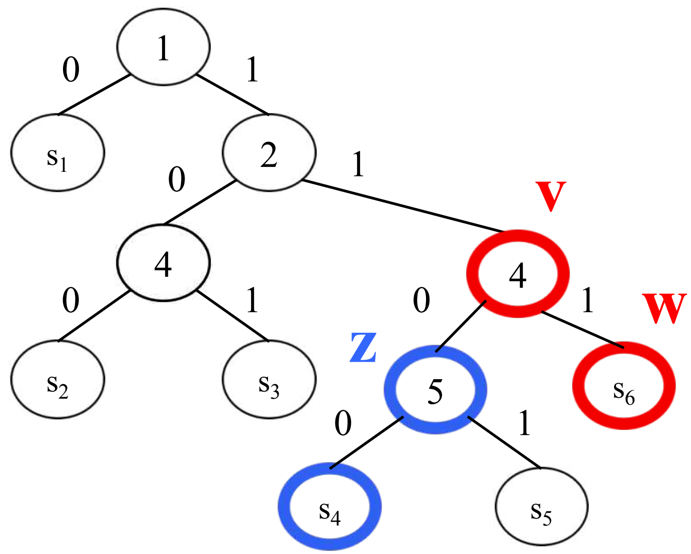


L = 3
k = 6
c = 5

O pai de **v** será o antigo pai de **z**.

Exemplo: inserção válida
Inserir chave 111101

s_1 = 0
s_2 = 1000
s_3 = 10010
s_4 = 11100
s_5 = 11101
s_6 = 111101



L = 3
k = 6
c = 5

Os filhos de **v** serão **w** e **z**, sendo **w** o filho esquerdo se d(L + 1**)** **= 0** ou o direito quando d(L + 1) **= 1**
Se **z** era a raiz de **T**, a nova raiz passa a ser **v**.

Exercícios

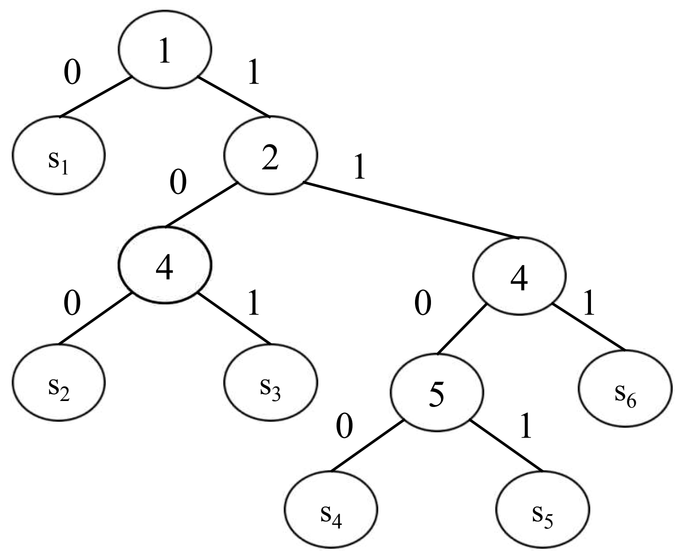
1. Desenhar a árvore digital correspondente ao conjunto de chaves abaixo

AR	ASA	RAS
ARA	ASAS	RASA
ARAR	ASSA	RASAS
ARARA	ASSAS	SA
ARAS	ASSAR	SAARA
ARRASA	ASSARA	SARA
ARRASAR	RA	SARAR
ARRASARA	RARA	SARARAS
AS	RARAS	SARAS

Exercícios

2. Simular o algoritmo de busca das chaves 11100, 111, 111101, 1010 na árvore Patrícia abaixo

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$
$s_6 = 111101$



Exercícios

3. Inserir as chaves 111, 101, 111110 na árvore Patrícia abaixo

$s_1 = 0$
$s_2 = 1000$
$s_3 = 10010$
$s_4 = 11100$
$s_5 = 11101$
$s_6 = 111101$