Universidade Federal de Ouro Preto
Campus João Monlevade

CSI 488 – Algoritmos e Estruturas de Dados I

TAD – Árvores Digitais

Prof. Mateus Ferreira Satler

Índice

1. Introdução

- Nos TADs vistos até o momento, são utilizadas **chaves** únicas para indexar as informações na estrutura.
  - **Exemplo**: um inteiro, número de matrícula, CPF, etc.

- As chaves usadas tem tamanho fixo.
  - Cabem em uma variável simples.

- Além disso, são tratadas como um único dado **indivisível** na estrutura.
  - Na busca por uma chave **X** qualquer, basta realizar uma comparação simples de **X** com as chaves armazenadas para retornar o resultado.

1. Introdução

- Na prática, isso nem sempre acontece.
  - E se as chaves excederem o espaço reservado?
    - Chaves dinâmicas.
  - Suponha que se deseje armazenar um texto e em seguida, tentar localizar as frases desse texto.
    - Neste caso, as chaves corresponderiam às frases do texto.

1. Introdução

- As soluções vistas até o momento não são aplicáveis para indexar esse tipo de chave.

- Neste cenário, as **Árvores Trie** e **Patricia** e a **Busca Digital** são apropriadas.

1.1. Busca Digital

- Na busca digital, a chave é tratada como um elemento **divisível.**
  - Cada chave é constituída de um **conjunto de caracteres** ou **dígitos.**
  - Na busca, a comparação é efetuada entre os dígitos que compõem as chaves (dígito a dígito, caractere a caractere).
- O método de pesquisa digital é análogo à pesquisa 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.

1.1. Busca Digital

- **Características das Chaves:**
  - Cada chave é formada por palavras sobre um **alfabeto** de símbolos.
  - Palavras com tamanho **VARIÁVEL** e **ILIMITADO.**
  - **Exemplos de alfabetos:**
    - {0,1}, {A, B, C,...Z, a, b, c,..., z}, {0,1,2,3,4,5,...,9}
  - **Exemplos de chaves:**
    - 0101010100000000001010000000001010
    - ABABBBABABA
    - Maria
    - 19034717

2. Árvores Trie

- O termo **trie** surgiu nos anos 60 e tem origem na palavra *retrieval*, uma vez que essa estrutura é usada basicamente na recuperação de dados.

- Na estrutura de dados trie as chaves são representadas caractere por caractere.

- Tries são usadas para fazer uma rápida busca em um texto grande.

2. Árvores Trie

- Cada chave é formada por uma combinação específica de símbolos do alfabeto.
  - As combinações são de comprimento variável e ilimitado.
  - Portanto, as chaves podem ser palavras, sequências binárias, códigos numéricos, etc.

2. Árvores Trie

- Em uma trie, as chaves são armazenadas e manipuladas de uma forma especial, pois são parcialmente compartilhadas entre os elementos.

- Ao invés de se comparar chaves inteiras entre si, as comparações são feitas componente a componente.

- Adicionalmente, as chaves são decompostas e as partes comuns entre elas são fundidas.

2. Árvores Trie

- As tries são boas para suportar tarefas de tratamento lexicográfico, tais como:
  - Manuseamento de dicionários.
  - Pesquisas em textos de grande dimensão.
  - Construção de índices de documentos.
  - Expressões regulares (padrões de pesquisa).

2. Árvores Trie

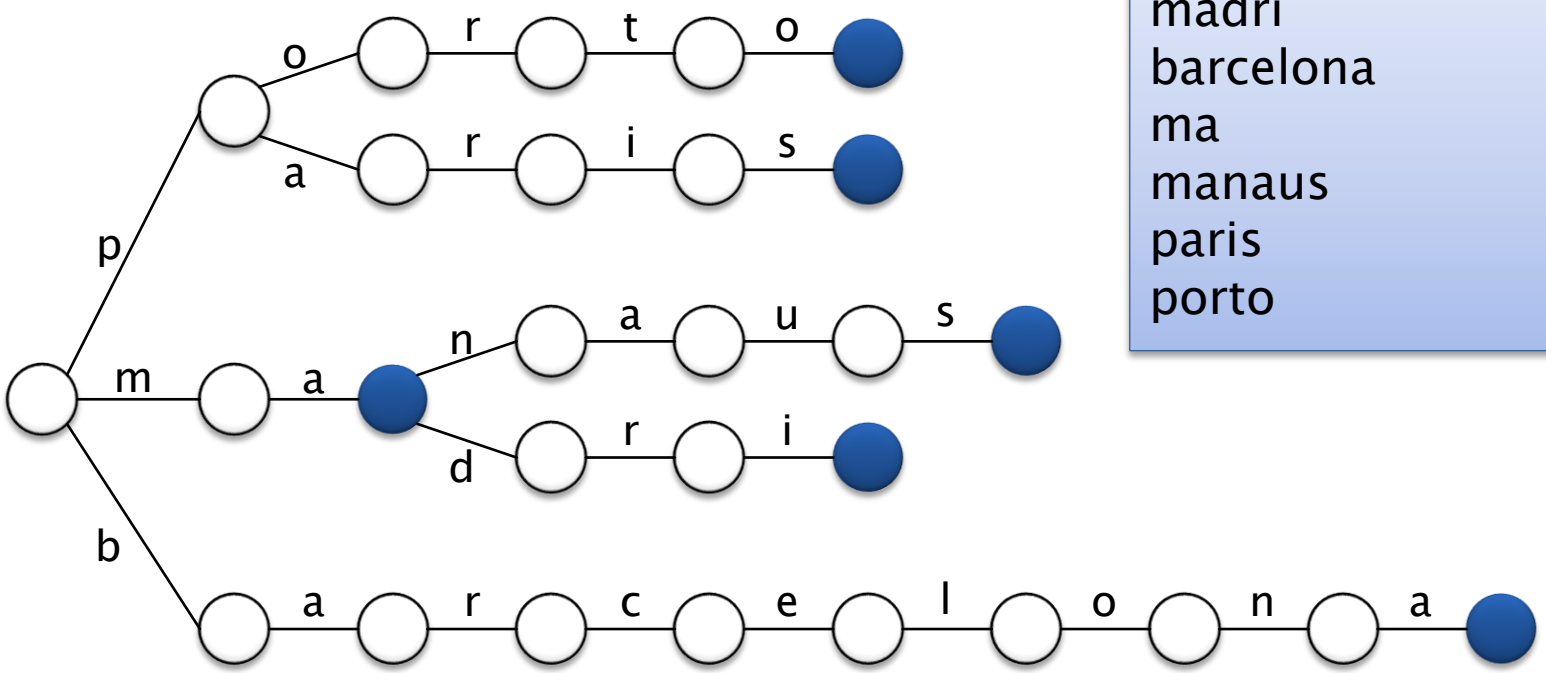
- **Características:**
  - Árvore **N-ária.**
  - Chaves em geral são caracteres.
  - Ao contrário da árvore binária de busca, nenhum nó armazena a chave.
    - Chave determinada pela posição na árvore.
  - O grau da árvore corresponde ao tamanho do alfabeto.

2. Árvores Trie

- **Características:**
  - Cada nível percorrido corresponde a avançar um digito na chave.
  - O caminho da raiz para qualquer nó é um **prefixo de uma chave**.
  - Descendentes do mesmo nó tem o mesmo prefixo.
  - Raiz representa a chave vazia.
  - Nós devem indicar quando completar uma chave.

2. Árvores Trie

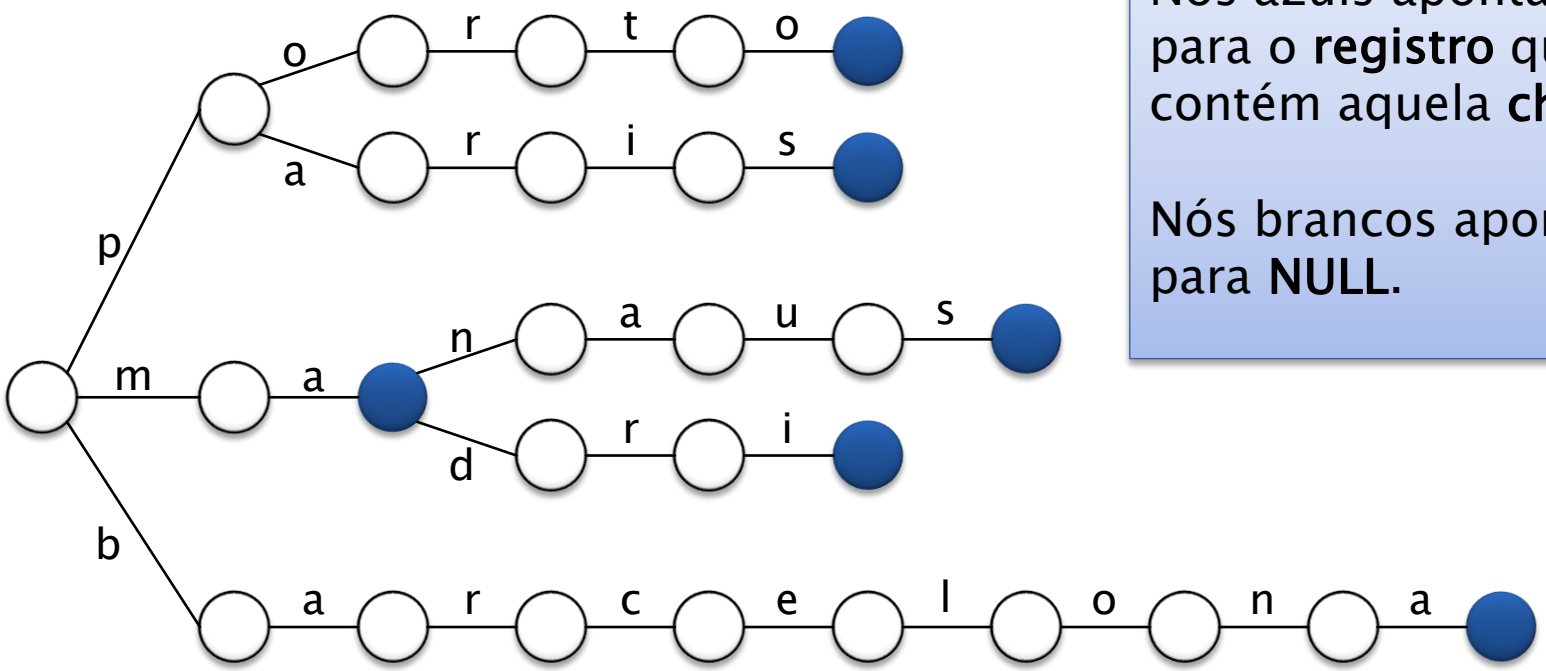
- **Exemplo:**



Chaves Indexadas:
madri
barcelona
ma
manaus
paris
porto

2. Árvores Trie

- **Exemplo:**



Nós azuis apontam para o **registro** que contém aquela **chave**.

Nós brancos apontam para **NULL**.

2. Árvores Trie

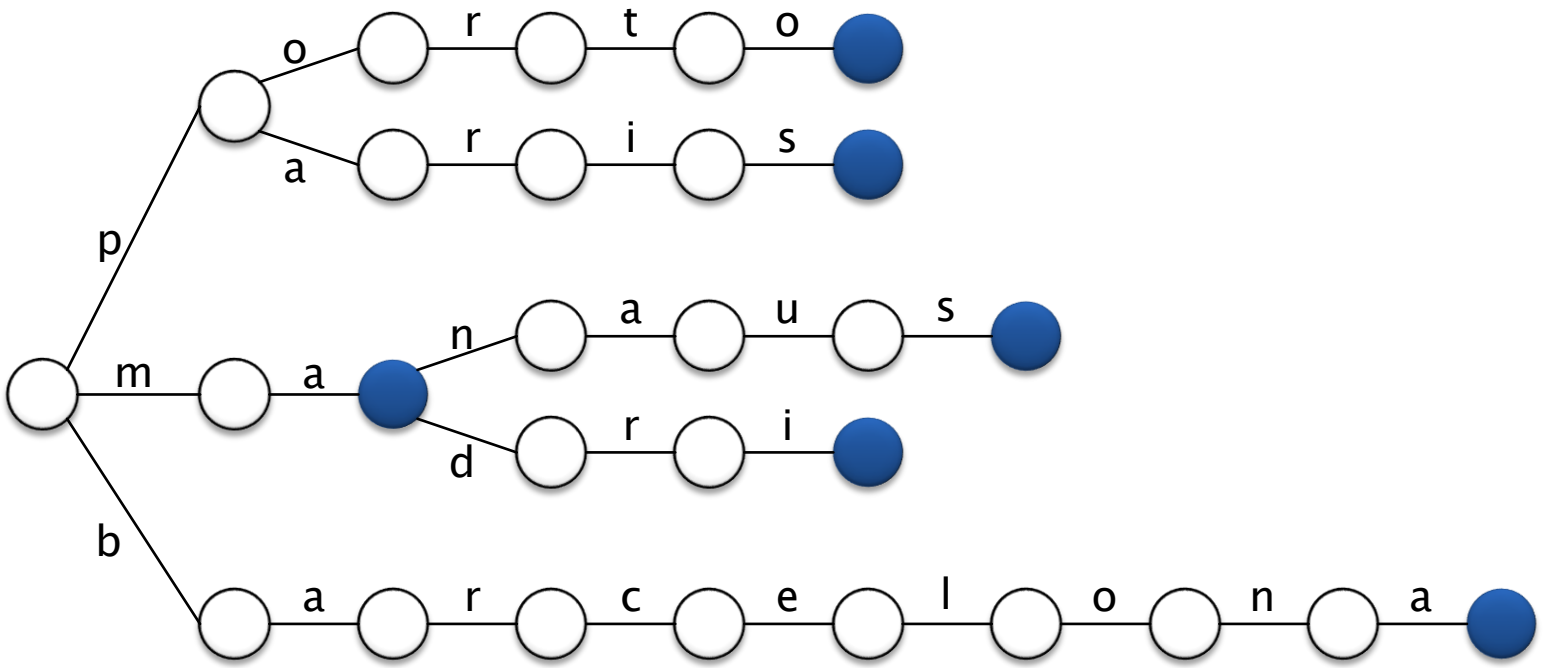
- **Definições:**
  - $S = \{s_1, \ldots, s_n\}$ é o conjunto de **chaves** a serem indexadas.
  - Cada chave s_i é formada por uma sequência de elementos d_j denominados **dígitos**.
  - Supõe-se que existe, em S, um total de **m** dígitos distintos, que compõe o alfabeto de S.
  - Os dígitos do alfabeto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

2. Árvores Trie

- **Definições:**
  - Uma árvore digital para **S** é uma árvore **m-ária T**, não vazia, tal que:
    - 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).
    - 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.

2. Árvores Trie

- **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}



2.1. Busca

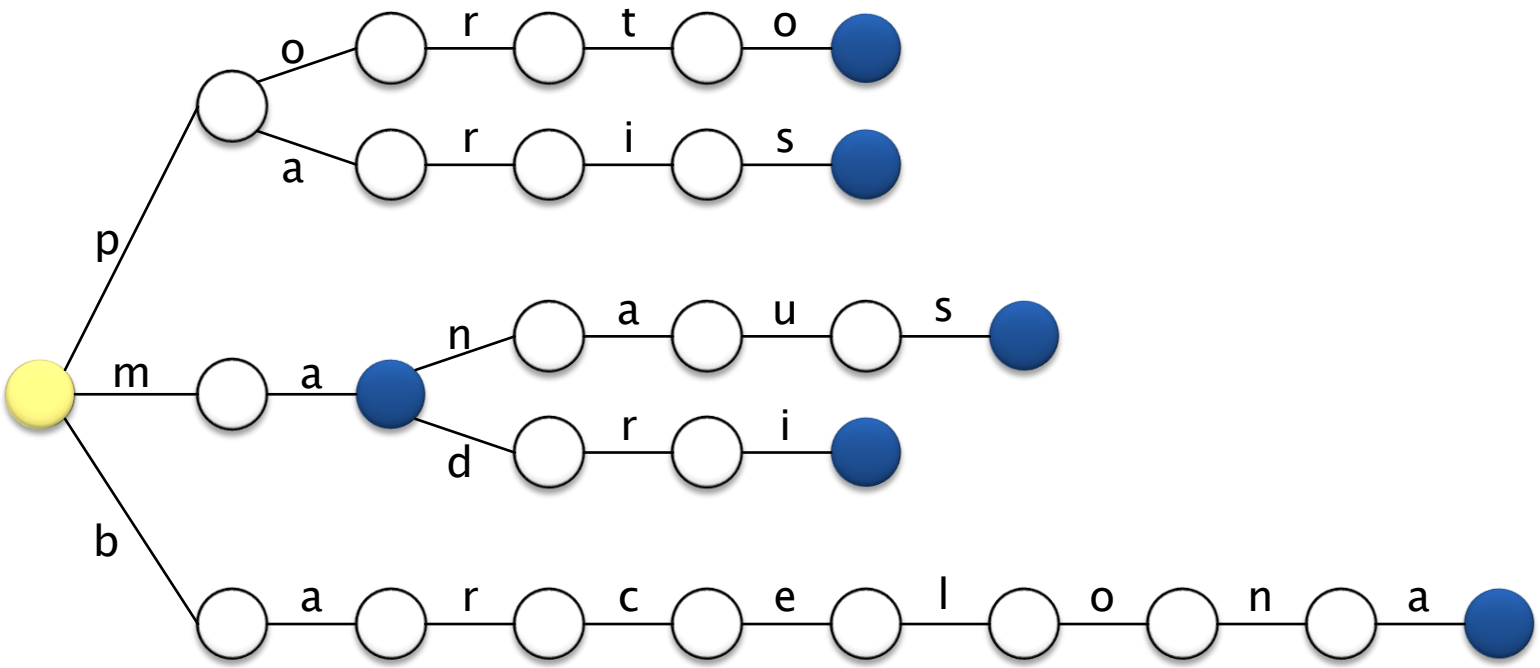
- Busca caractere a caractere a CHAVE.

1. Se o caractere não pertence a árvore a chave não pertence a trie.

2. Se o caractere pertence a árvore, verifique o próximo e caso todos os caracteres pertençam em sequencia a trie, a chave pertence a árvore.

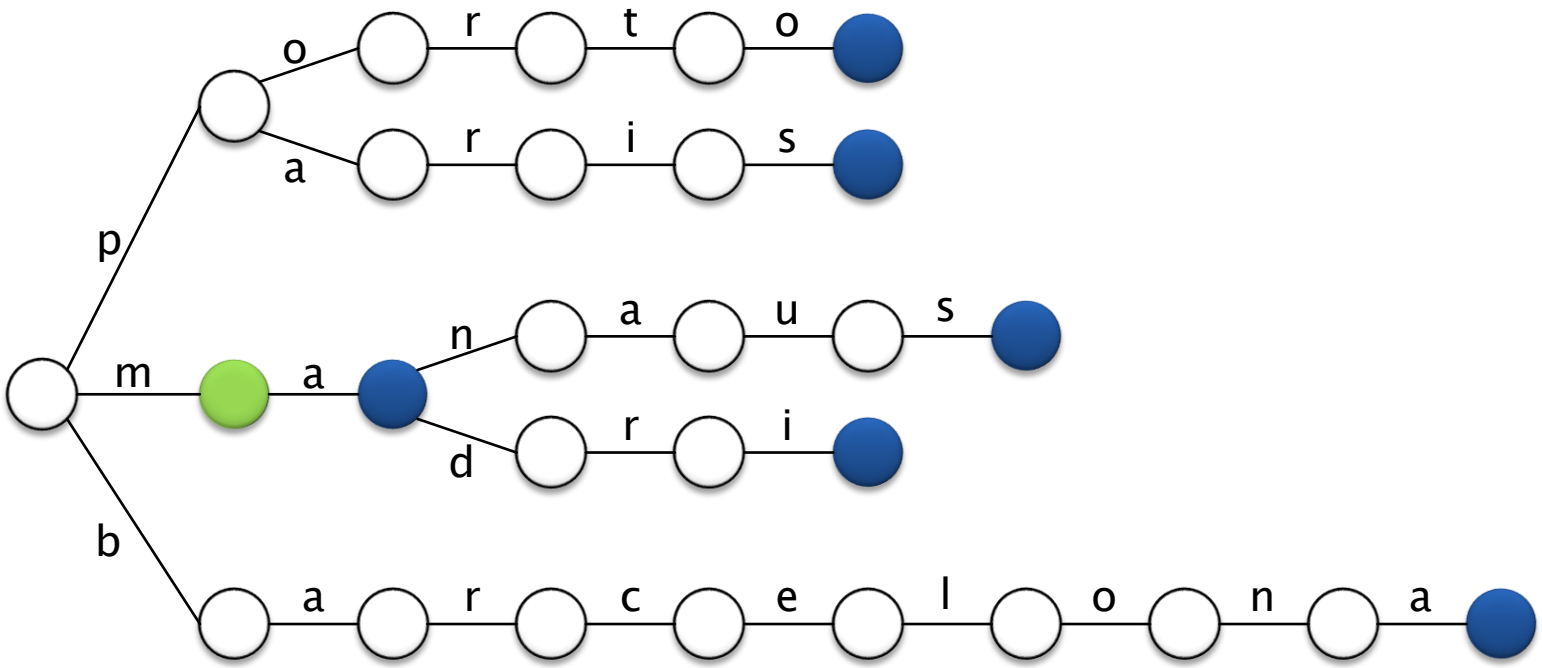
2.1. Busca

- **Exemplo 1**: buscar a chave **madri**



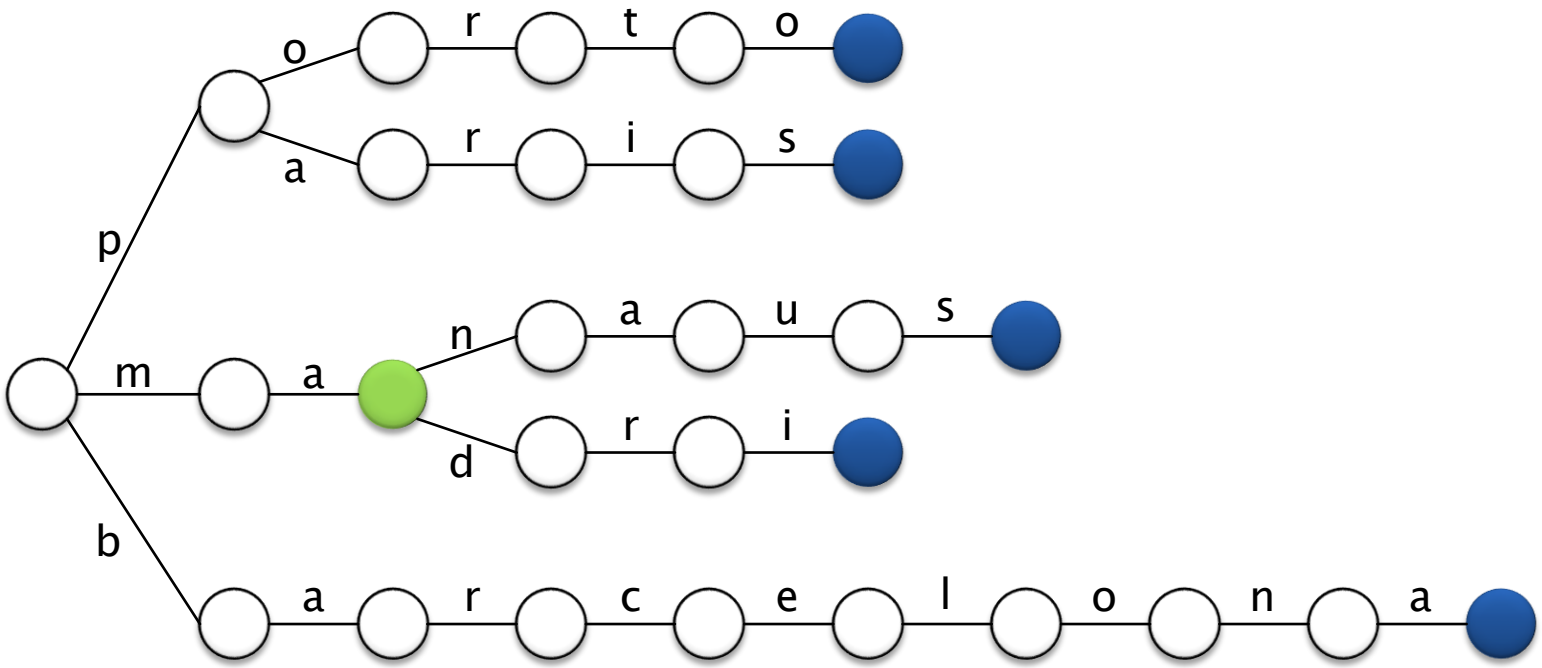
2.1. Busca

- **Exemplo 1**: buscar a chave **madri**



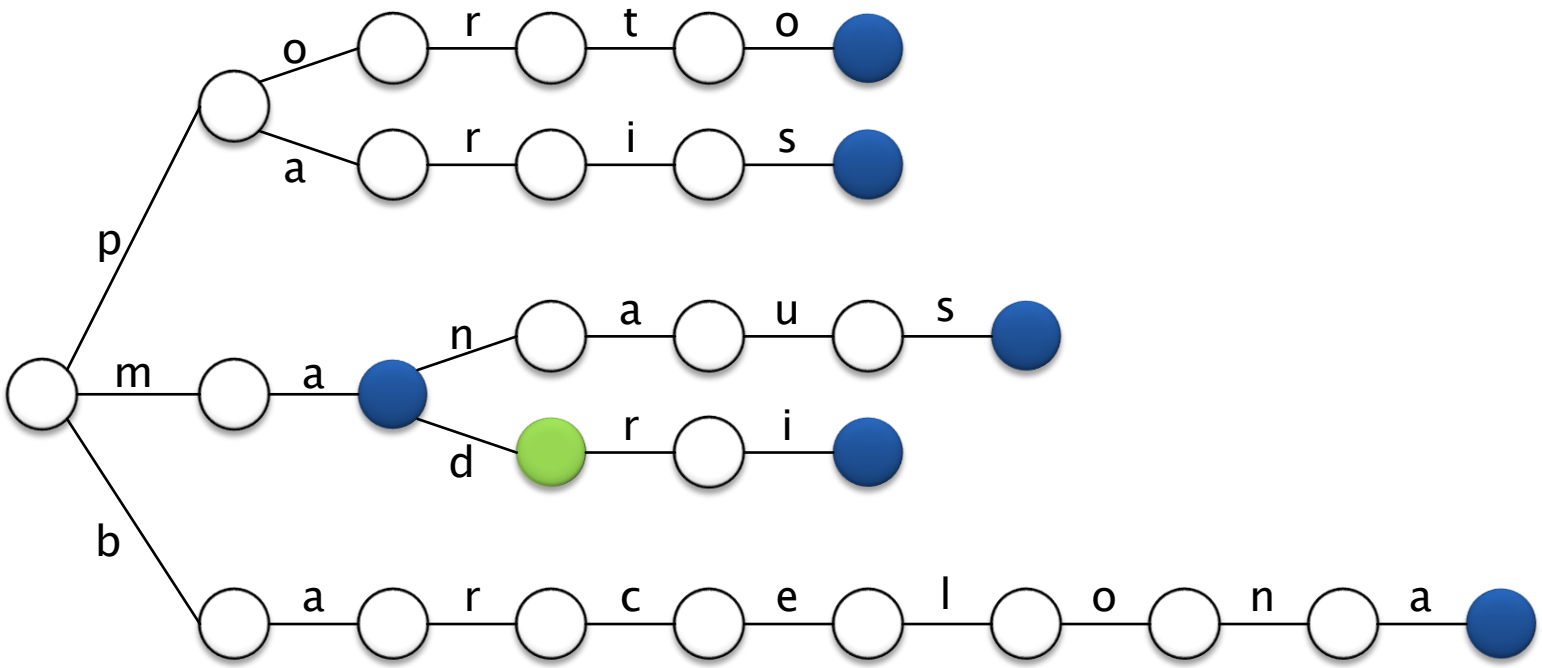
2.1. Busca

- **Exemplo 1**: buscar a chave **madri**



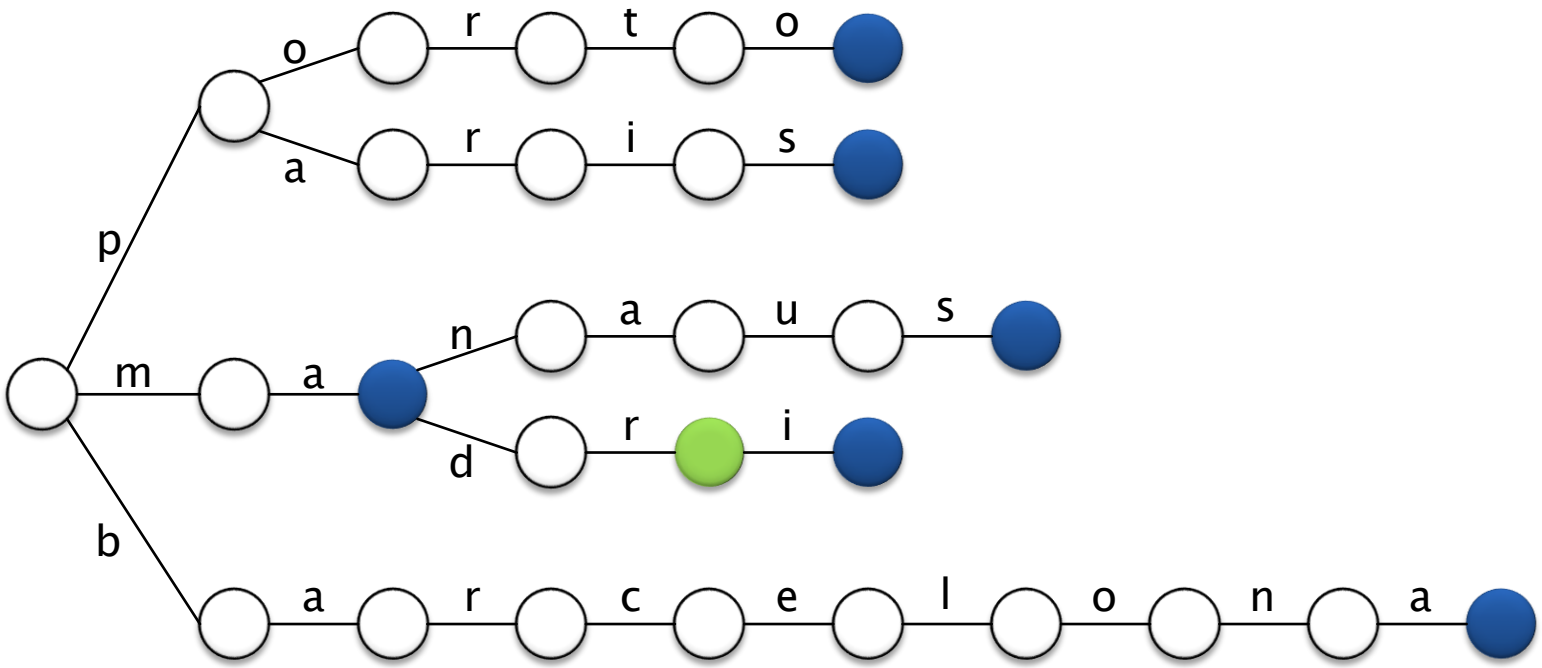
2.1. Busca

- **Exemplo 1**: buscar a chave **madri**



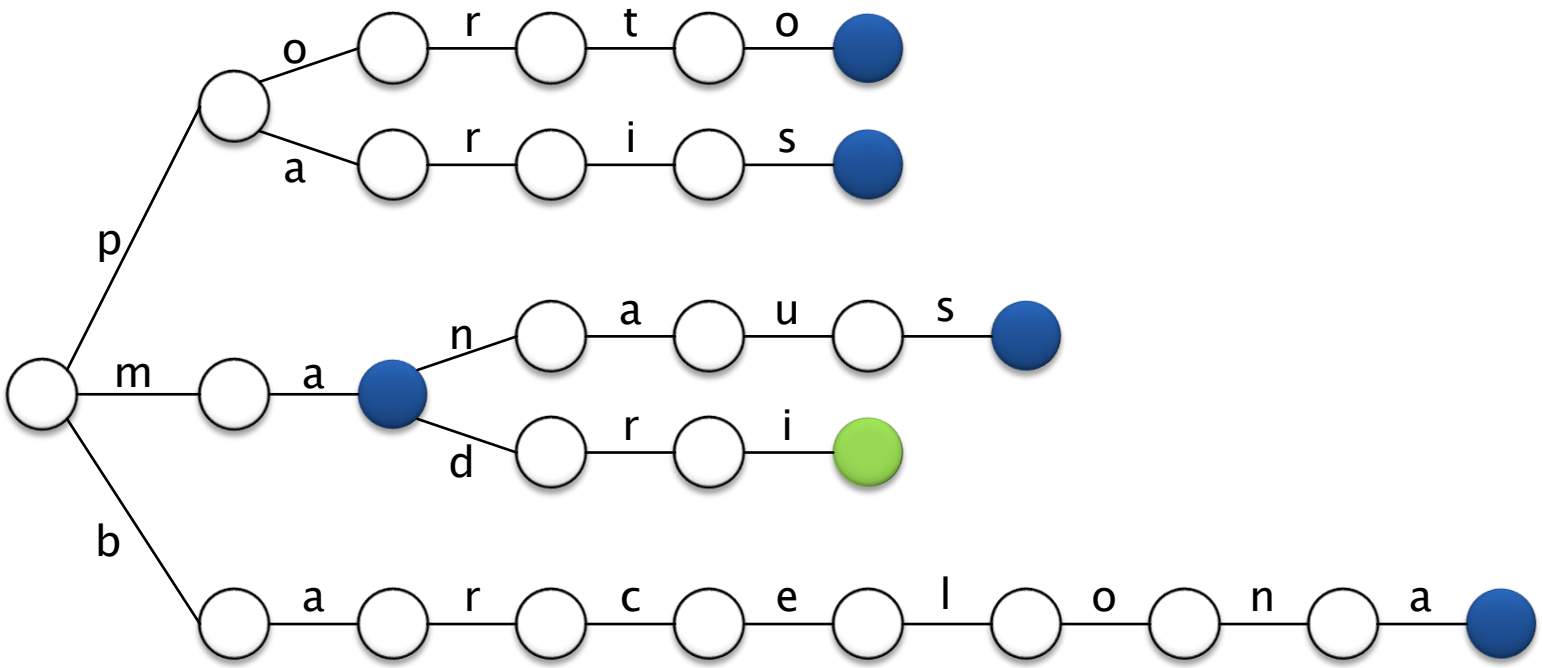
2.1. Busca

- **Exemplo 1**: buscar a chave **madri**



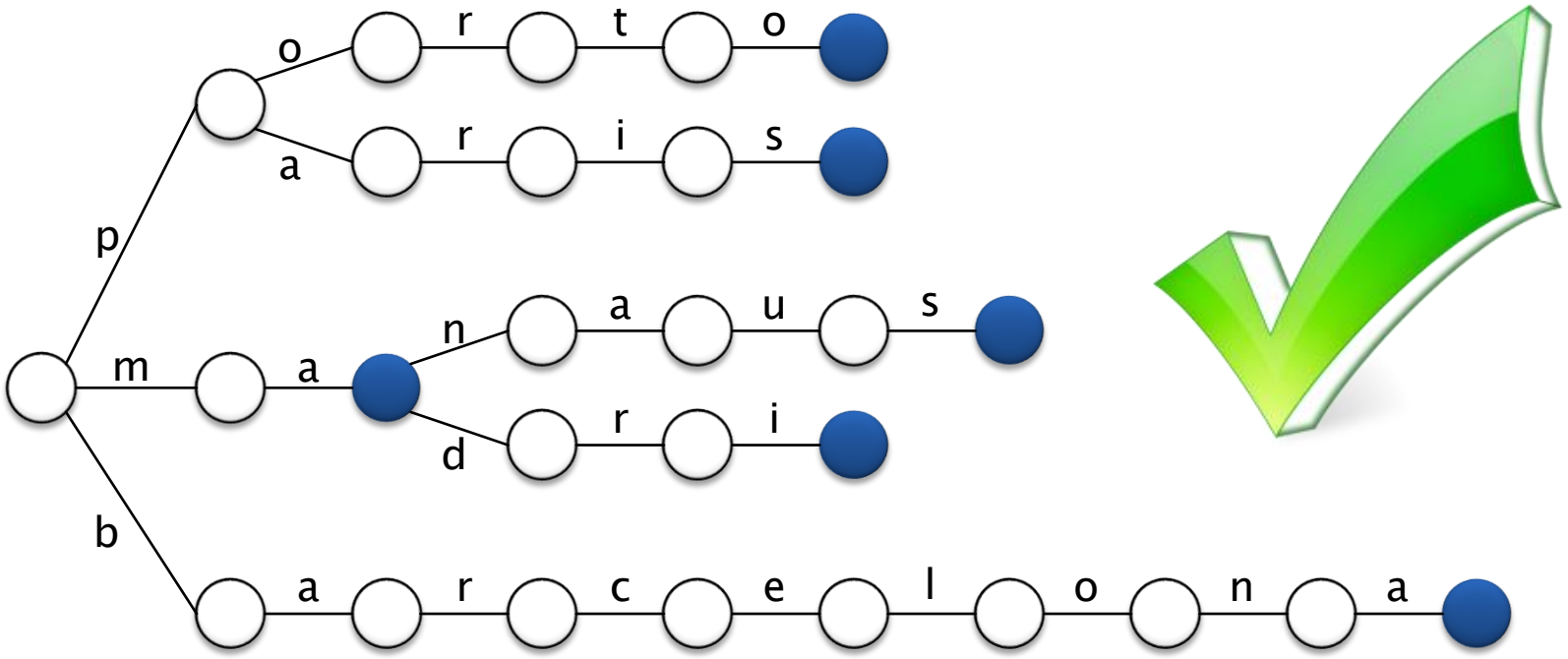
2.1. Busca

- **Exemplo 1**: buscar a chave **madri**



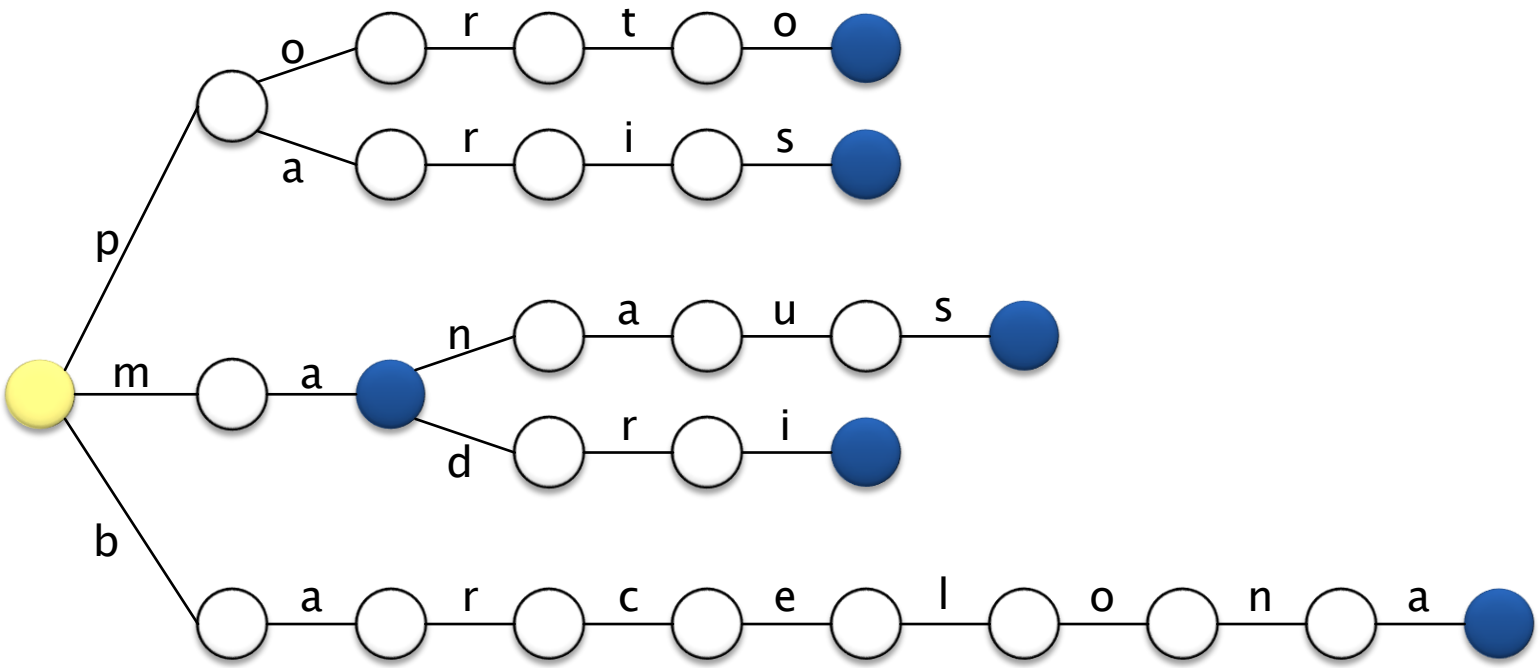
2.1. Busca

- **Exemplo 1**: buscar a chave **madri**



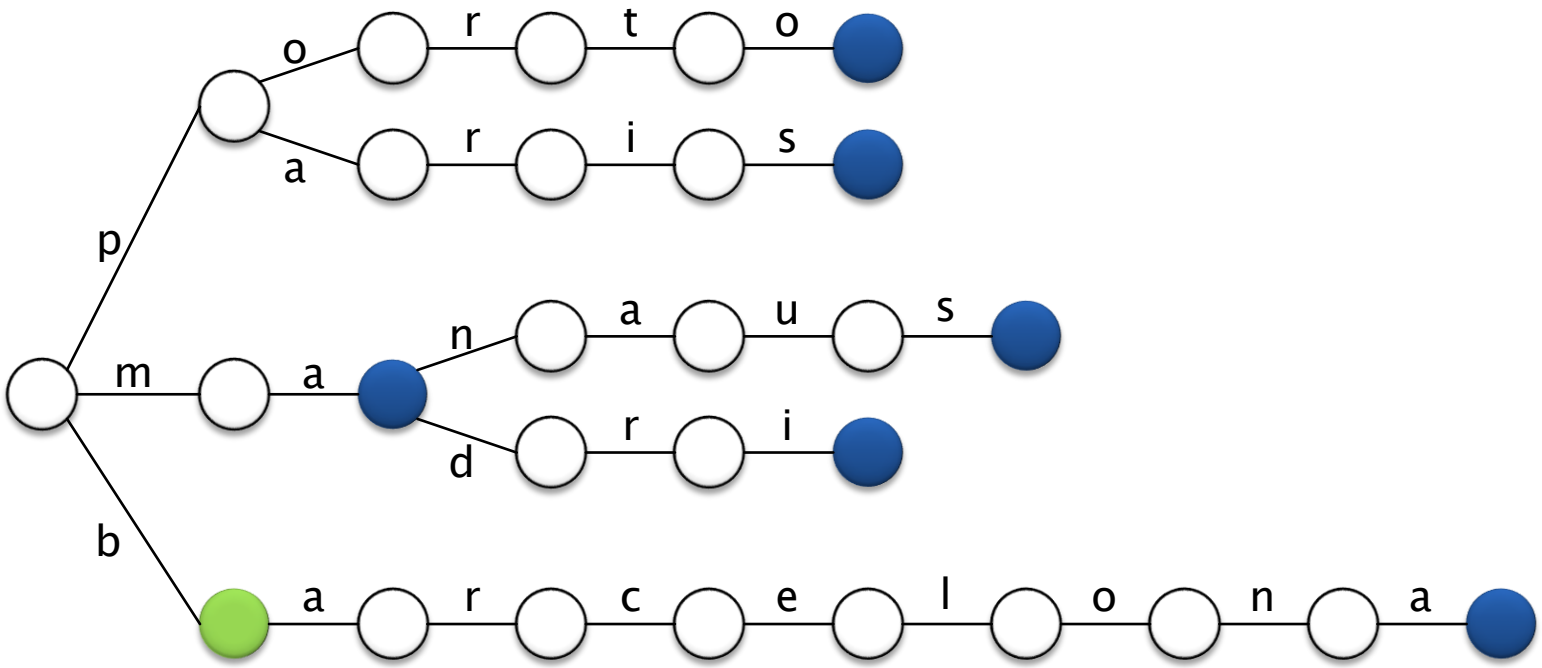
2.1. Busca

- **Exemplo 2**: buscar a chave **bahia**



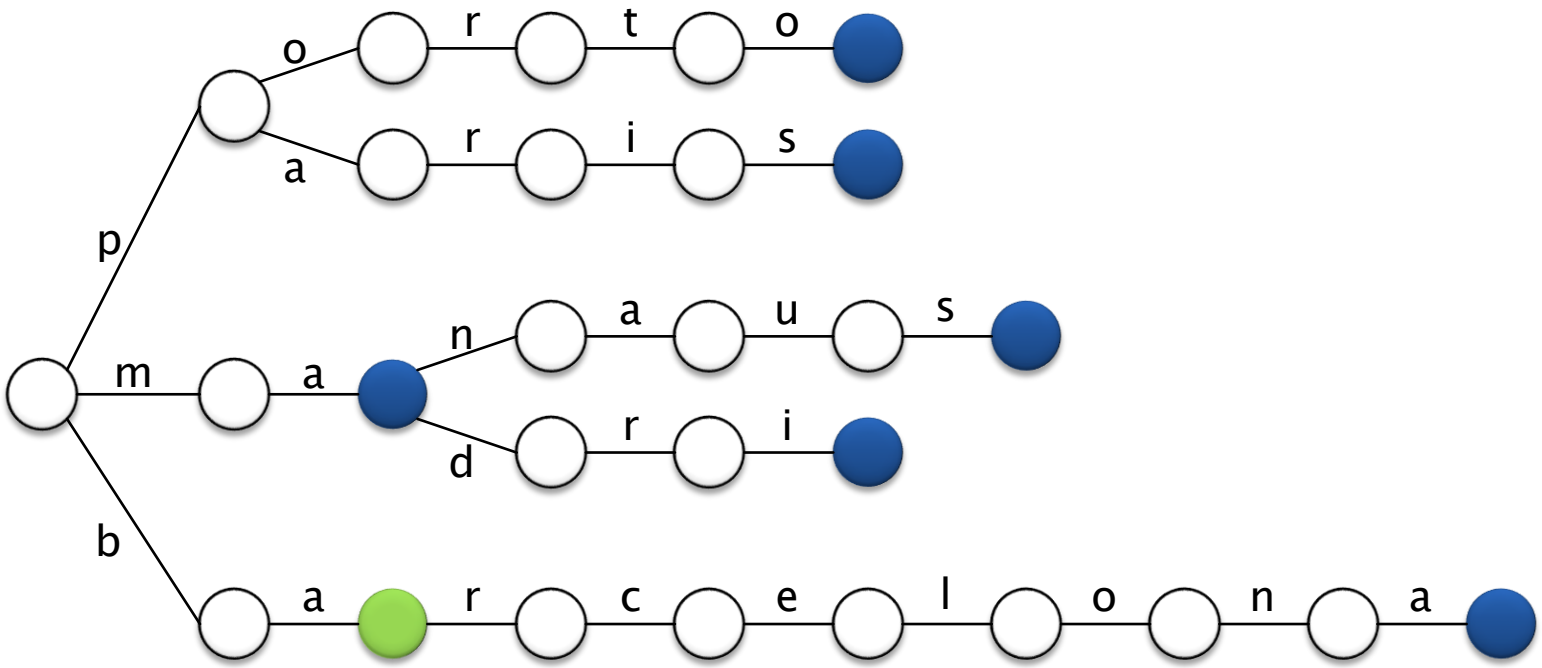
2.1. Busca

- **Exemplo 2**: buscar a chave **bahia**



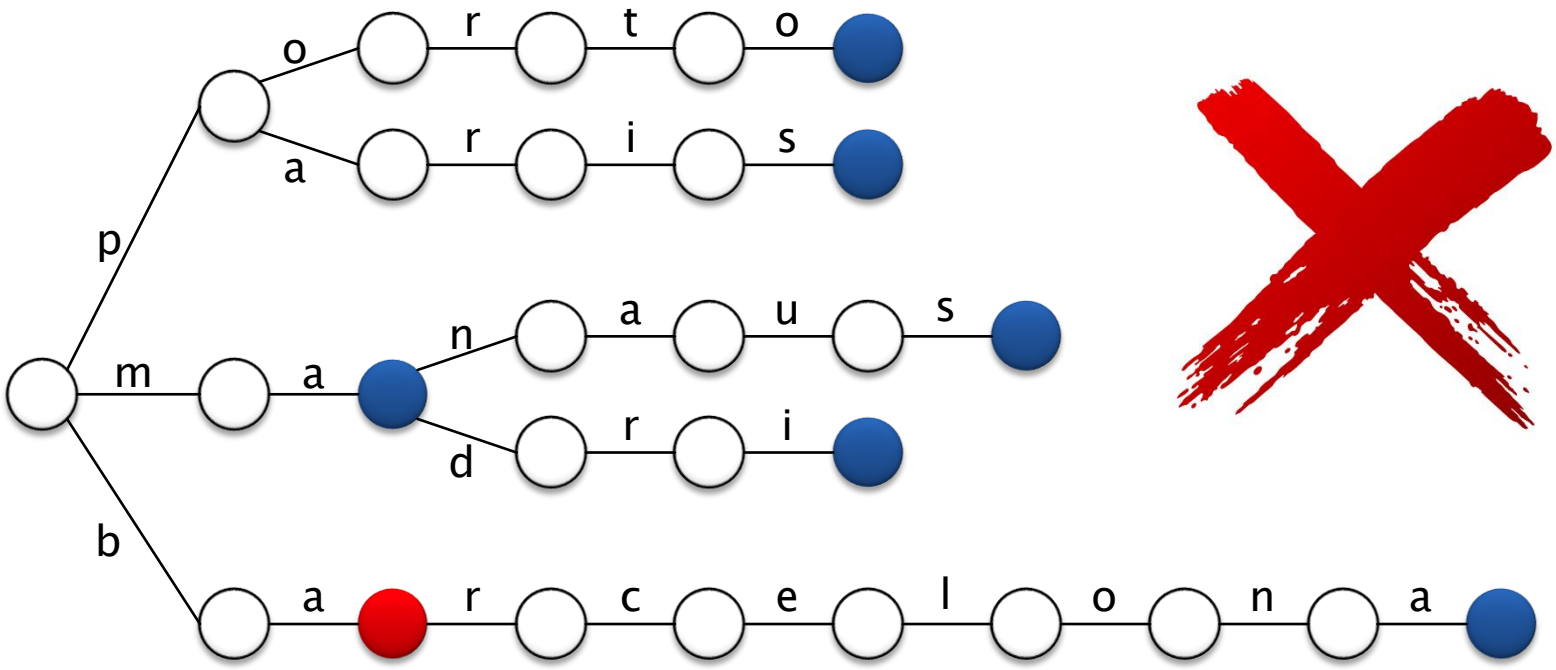
2.1. Busca

- **Exemplo 2**: buscar a chave **bahia**



2.1. Busca

- **Exemplo 2**: buscar a chave **bahia**



2.2. Inserção

- **Sequência de passos:**

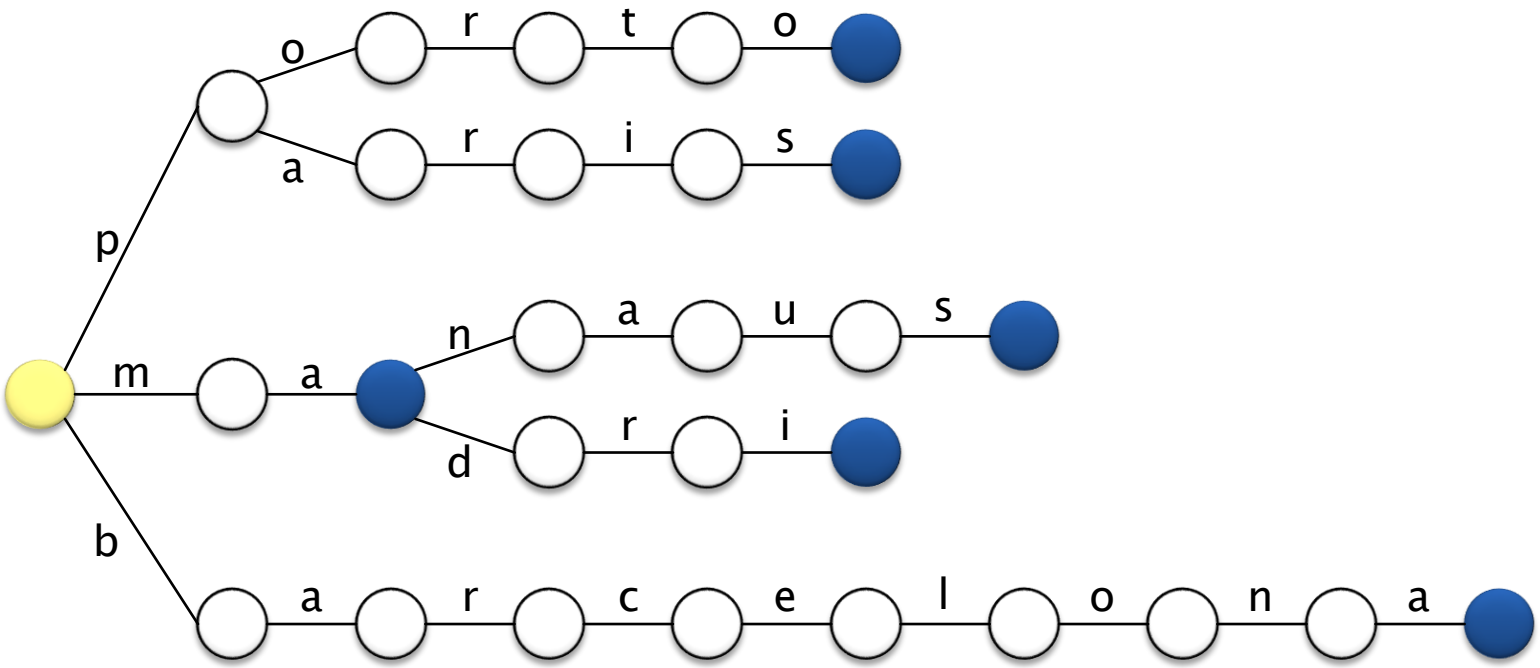
1. Faz-se uma busca pela palavra a ser inserida.
   - Se ela já existir na trie nada é feito.

2. Caso contrário, é recuperado o último nó **n** da maior sub-string da palavra a ser inserida.

3. O restante dos caracteres da chave são adicionados na trie a partir do nó **n**.

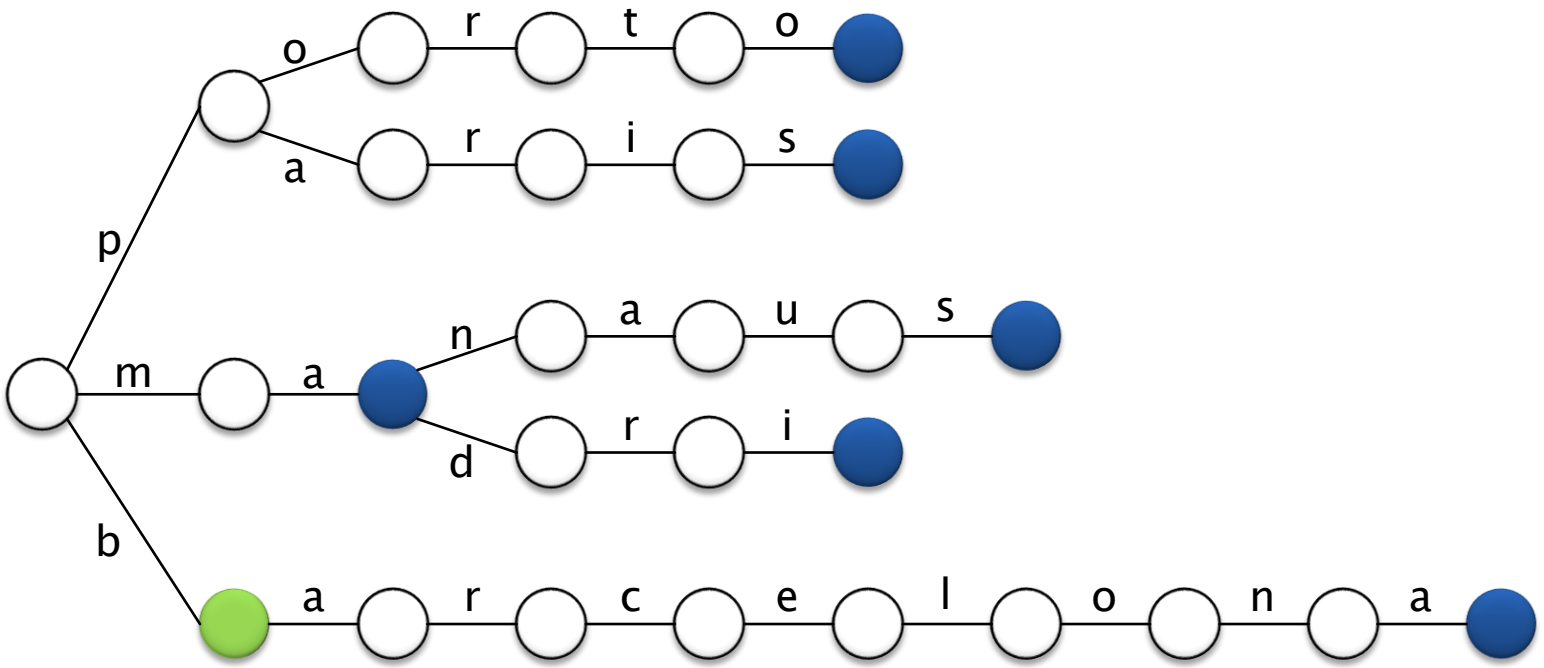
2.2. Inserção

- **Exemplo:** inserir a chave **bahia**



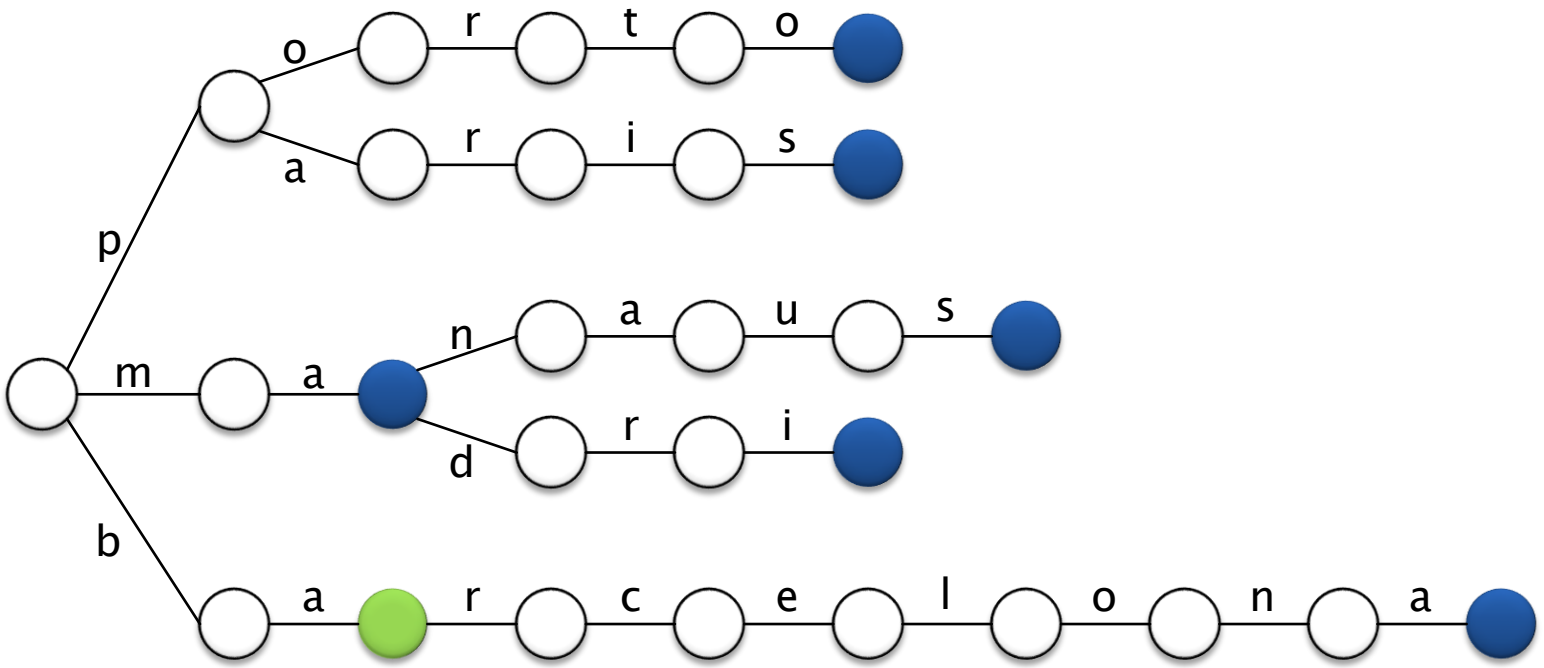
2.2. Inserção

- **Exemplo:** inserir a chave **bahia**



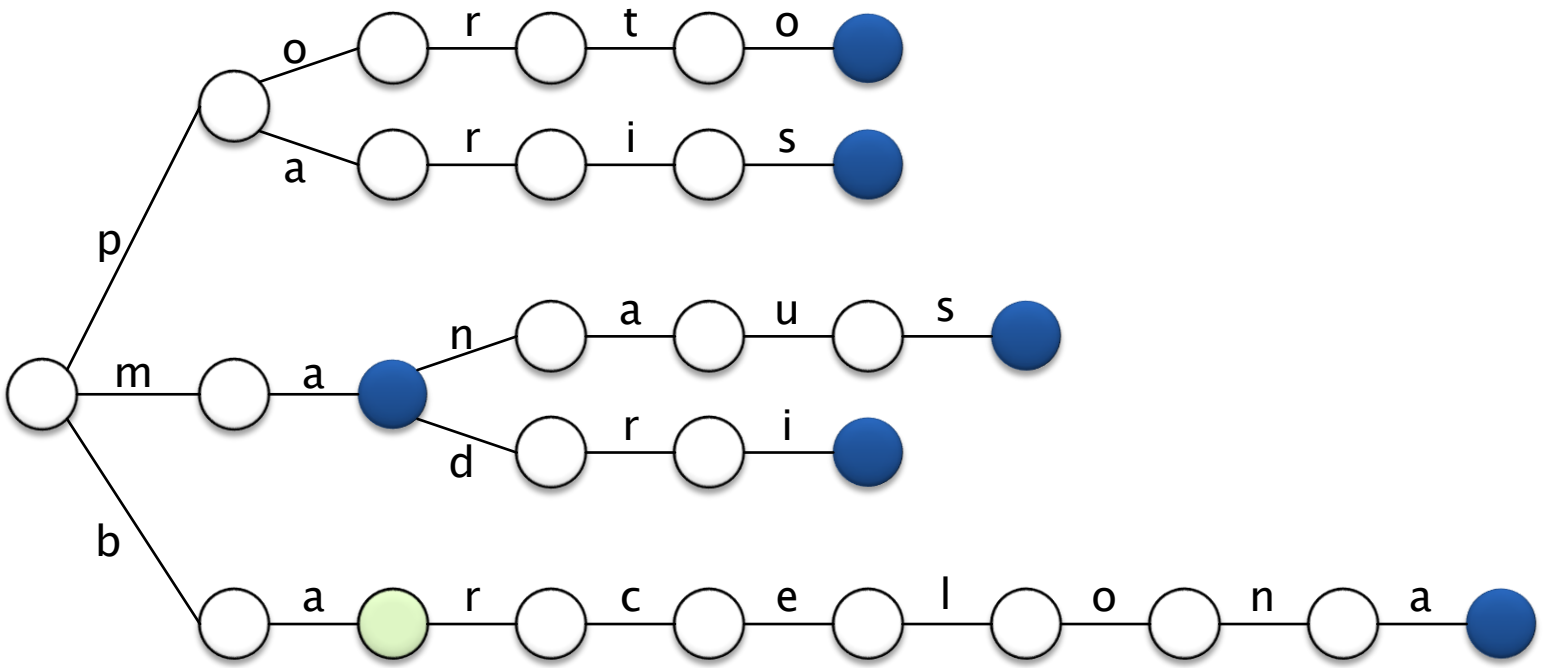
2.2. Inserção

- **Exemplo:** inserir a chave **bahia**



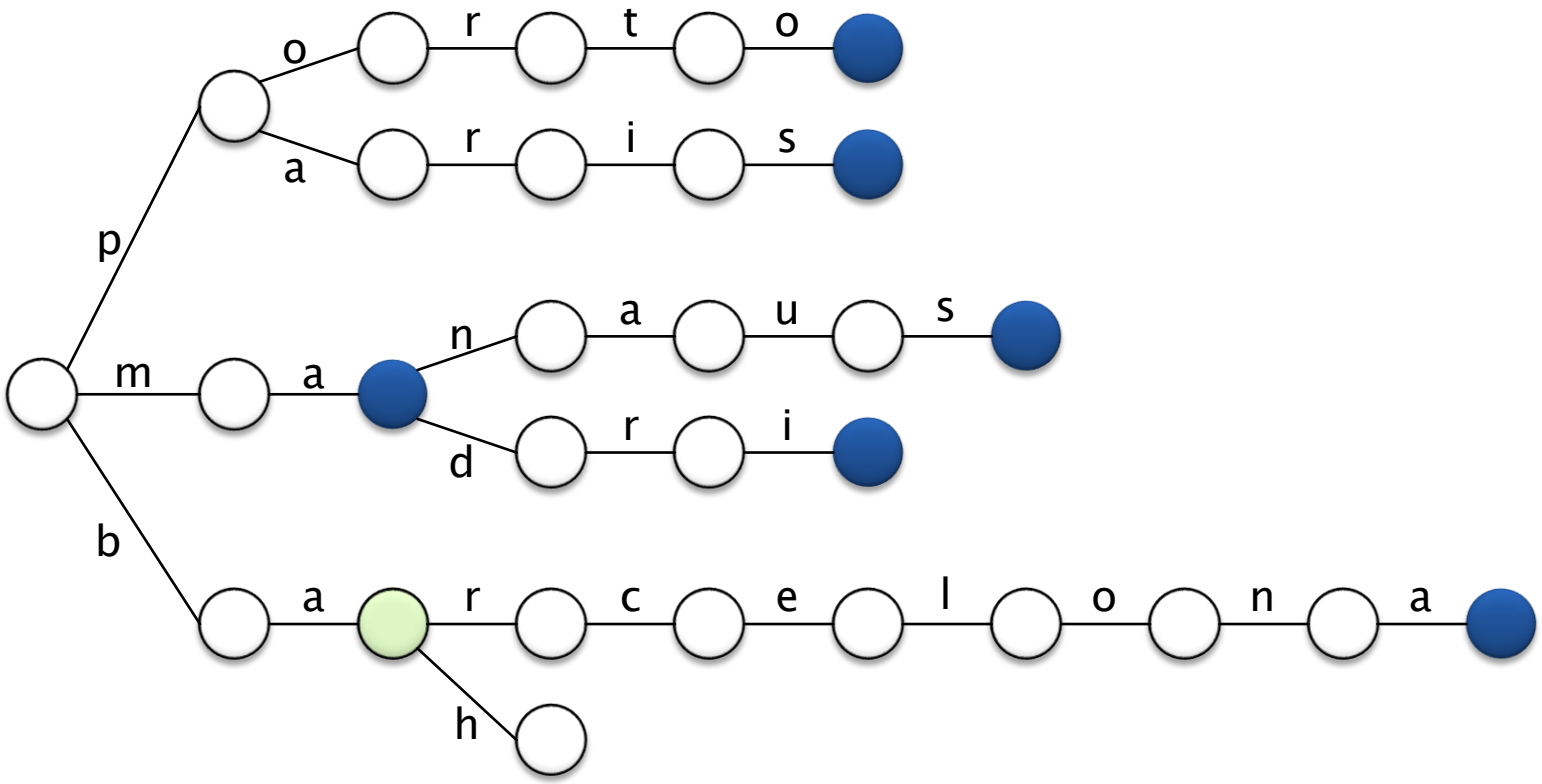
2.2. Inserção

- **Exemplo:** inserir a chave **bahia**



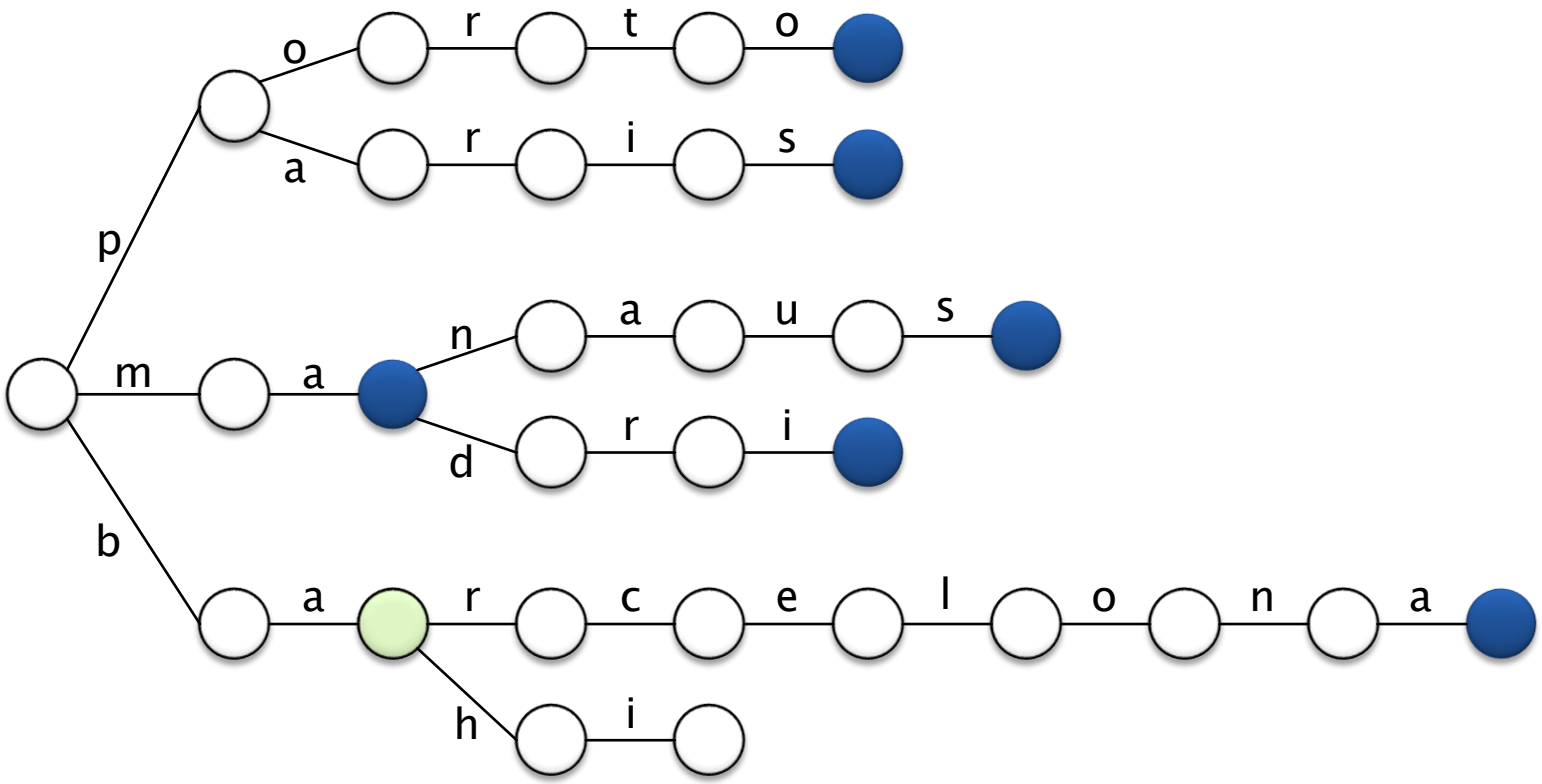
2.2. Inserção

- **Exemplo:** inserir a chave **bahia**



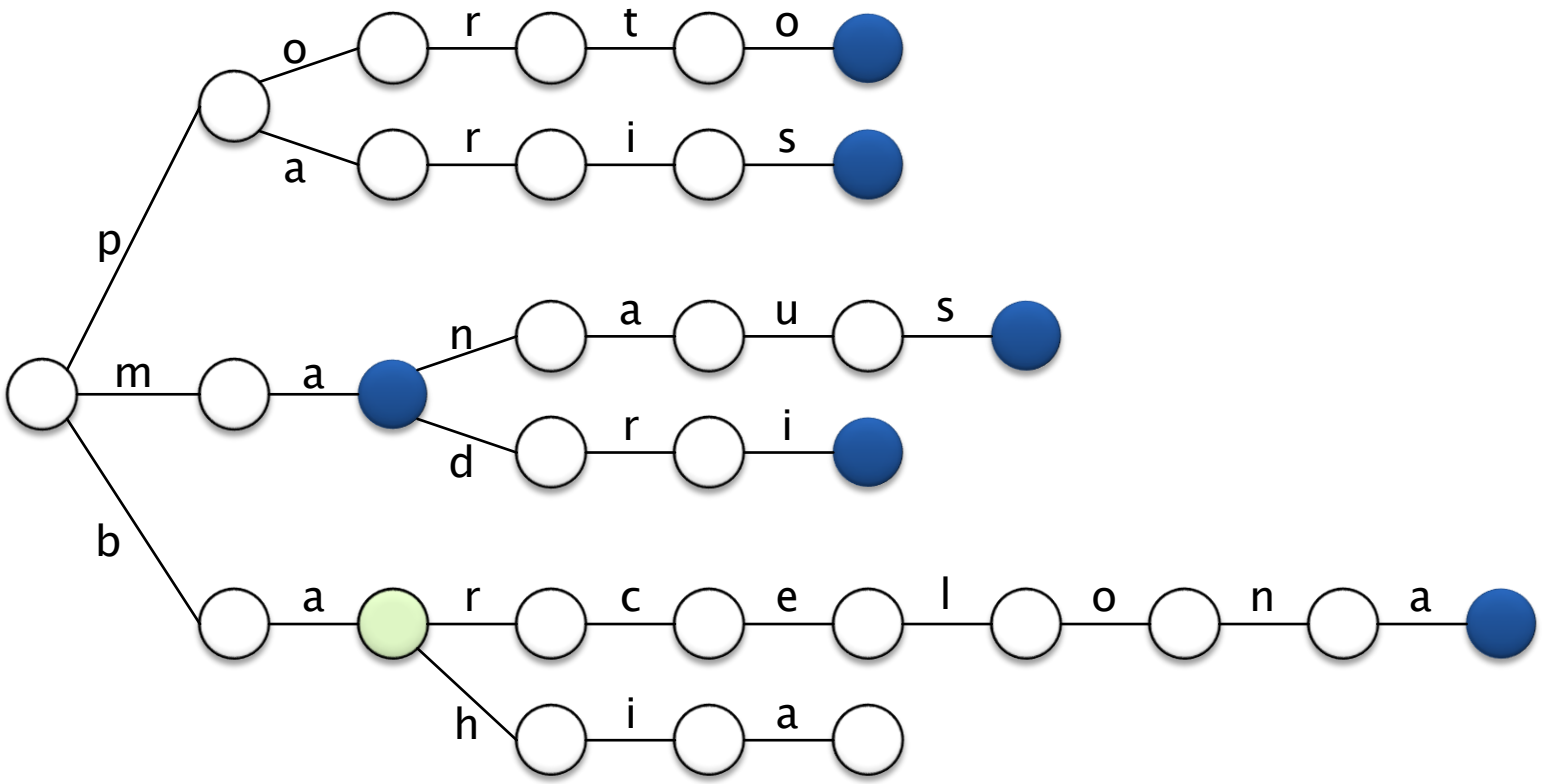
2.2. Inserção

- **Exemplo:** inserir a chave **bahia**



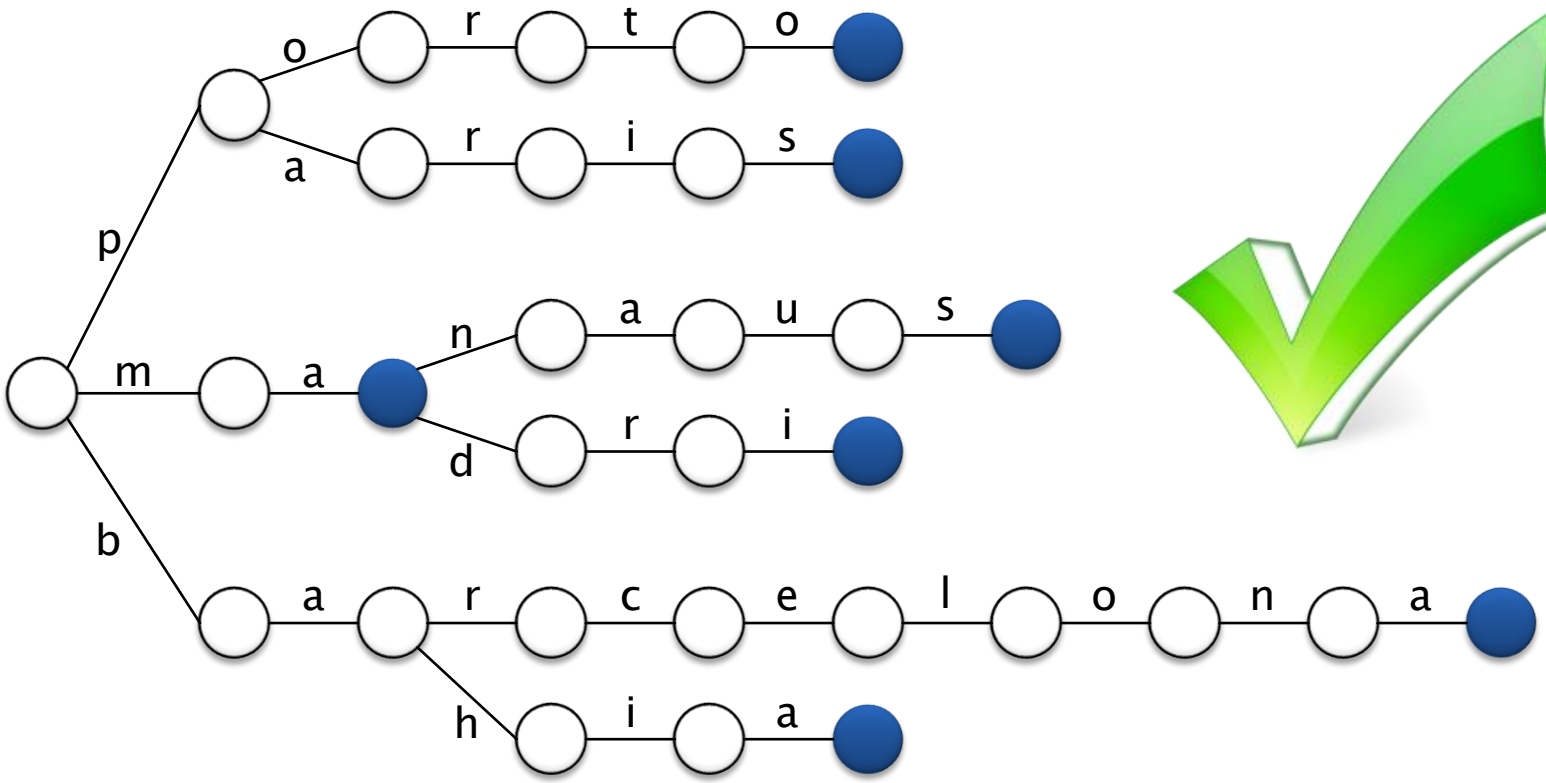
2.2. Inserção

- **Exemplo:** inserir a chave **bahia**



2.2. Inserção

- **Exemplo:** inserir a chave **bahia**



2.3. Remoção

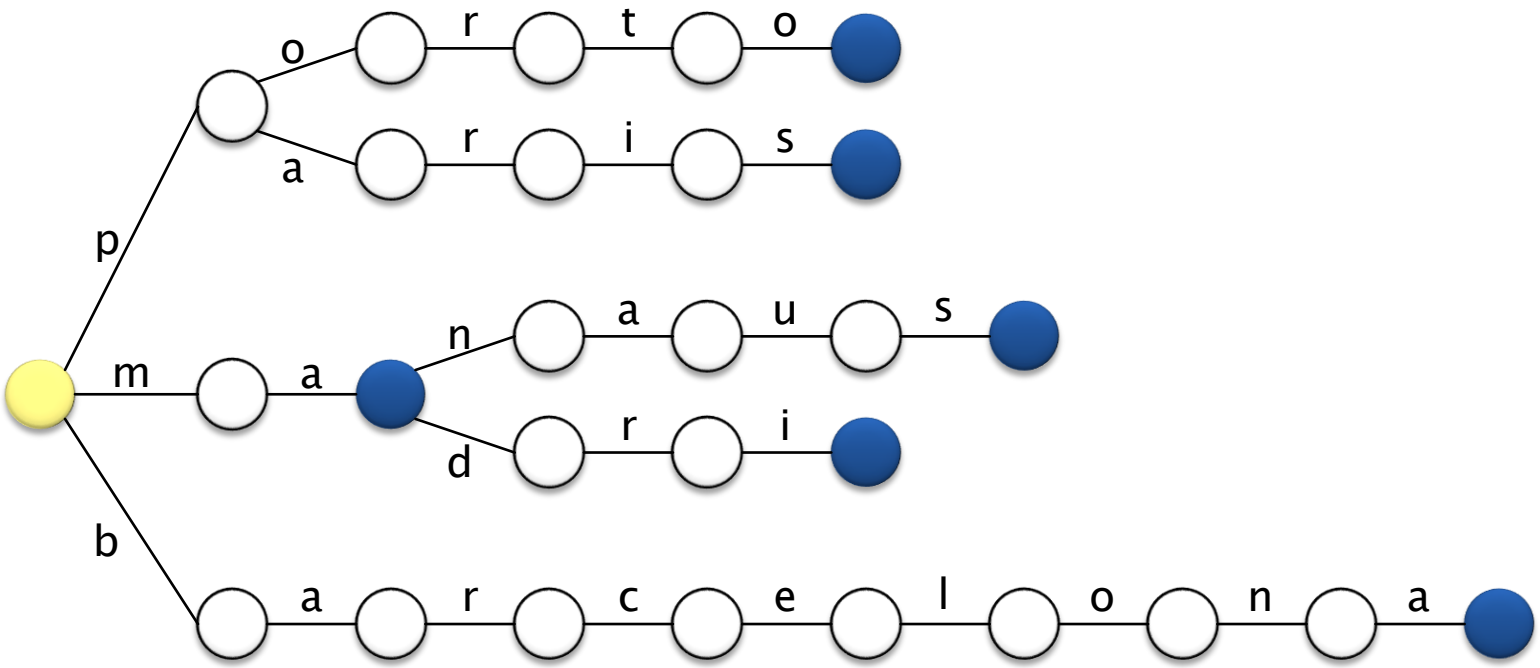
- **Sequência de passos:**

1. Busca-se a chave a ser removida.

2. A partir da folha (buttom-up), são removidos todos os nós que tem **apenas um filho.**

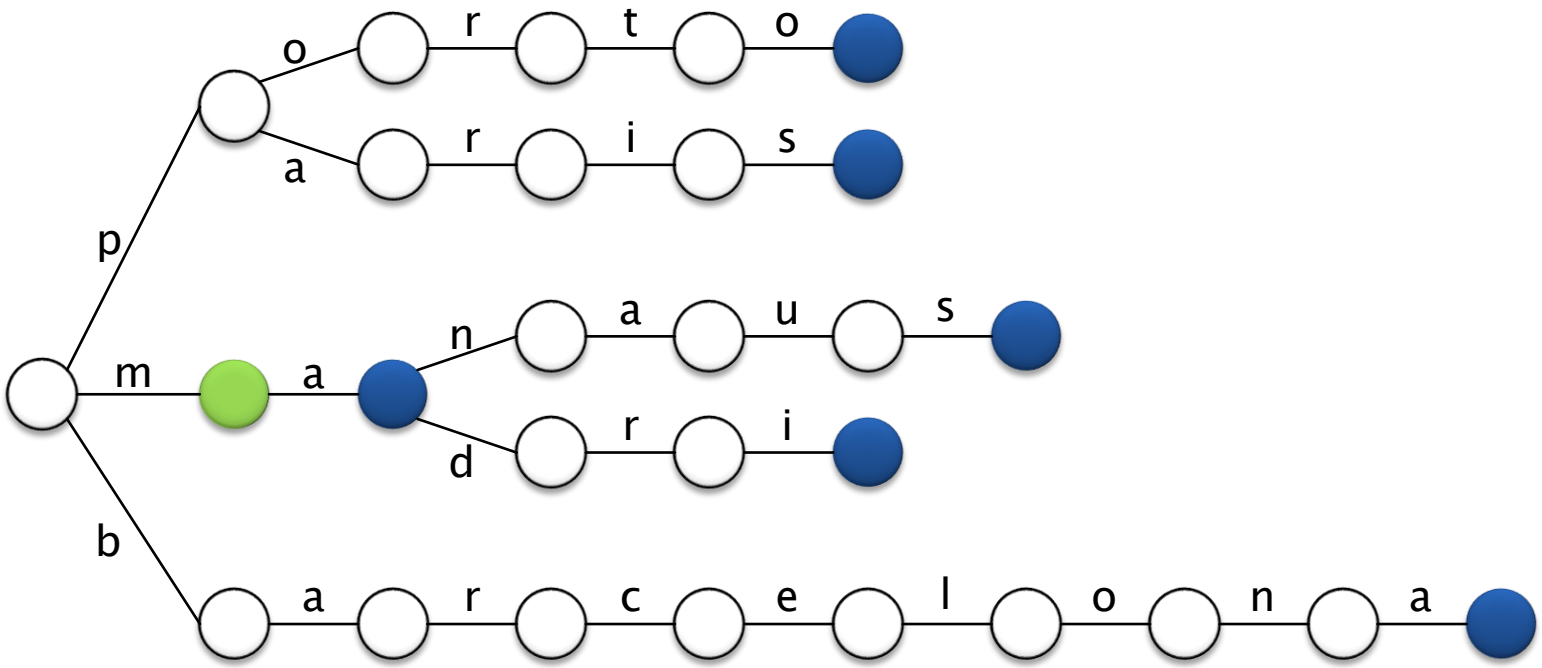
2.3. Remoção

- **Exemplo**: remover a chave **madri**



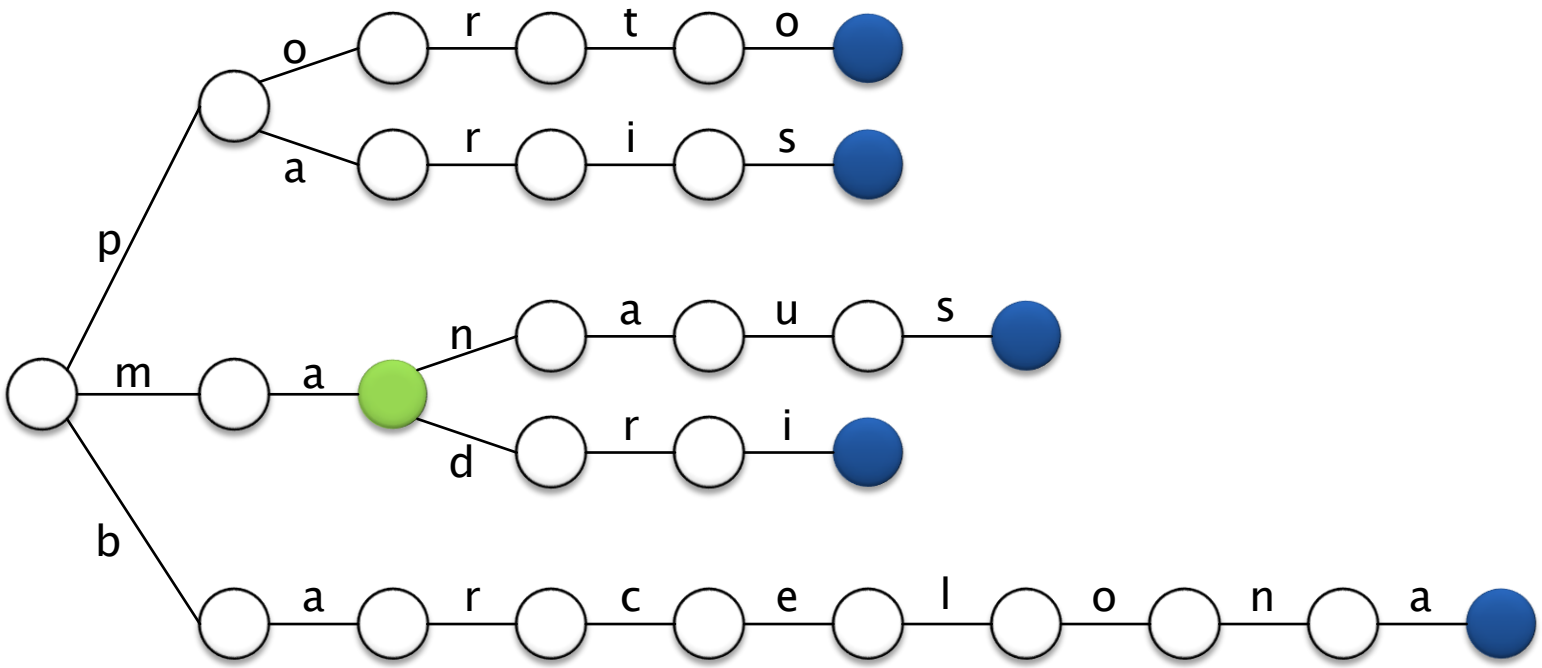
2.3. Remoção

- **Exemplo**: remover a chave **madri**



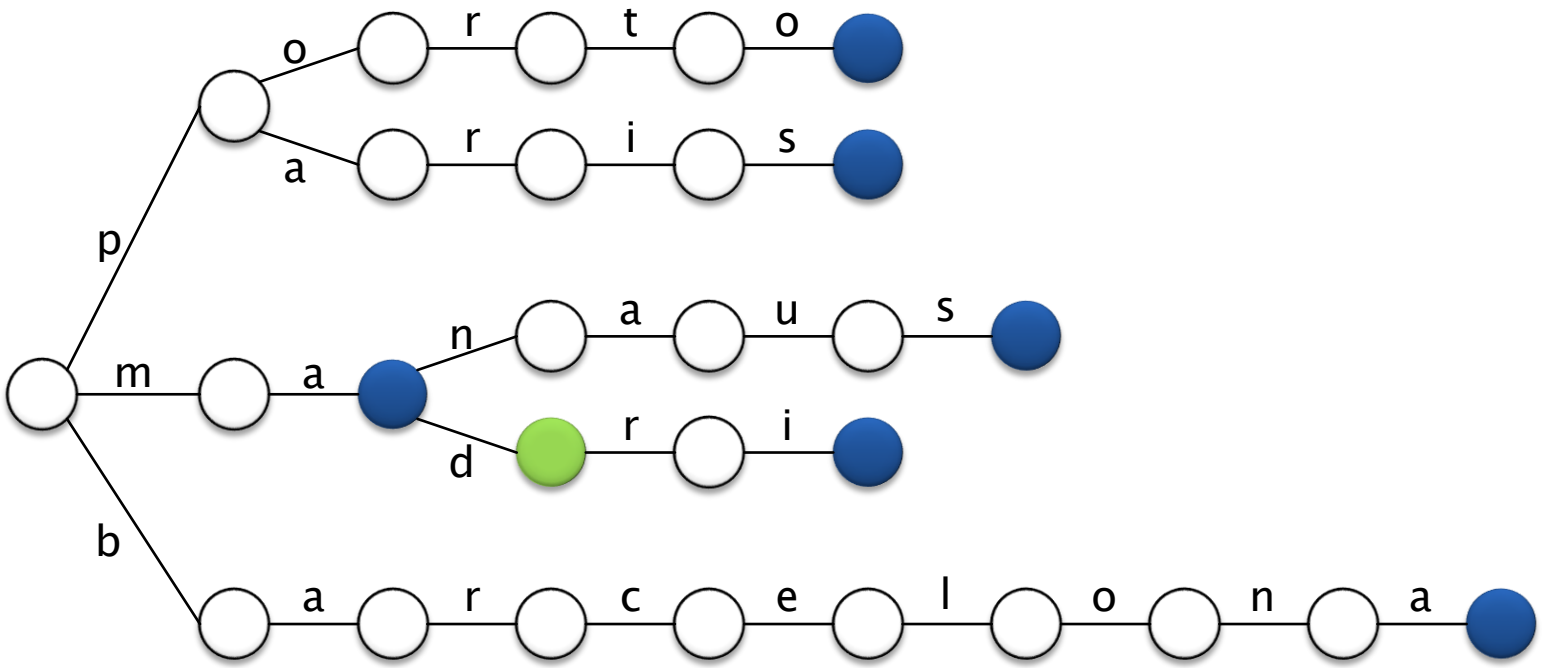
2.3. Remoção

- **Exemplo**: remover a chave **madri**



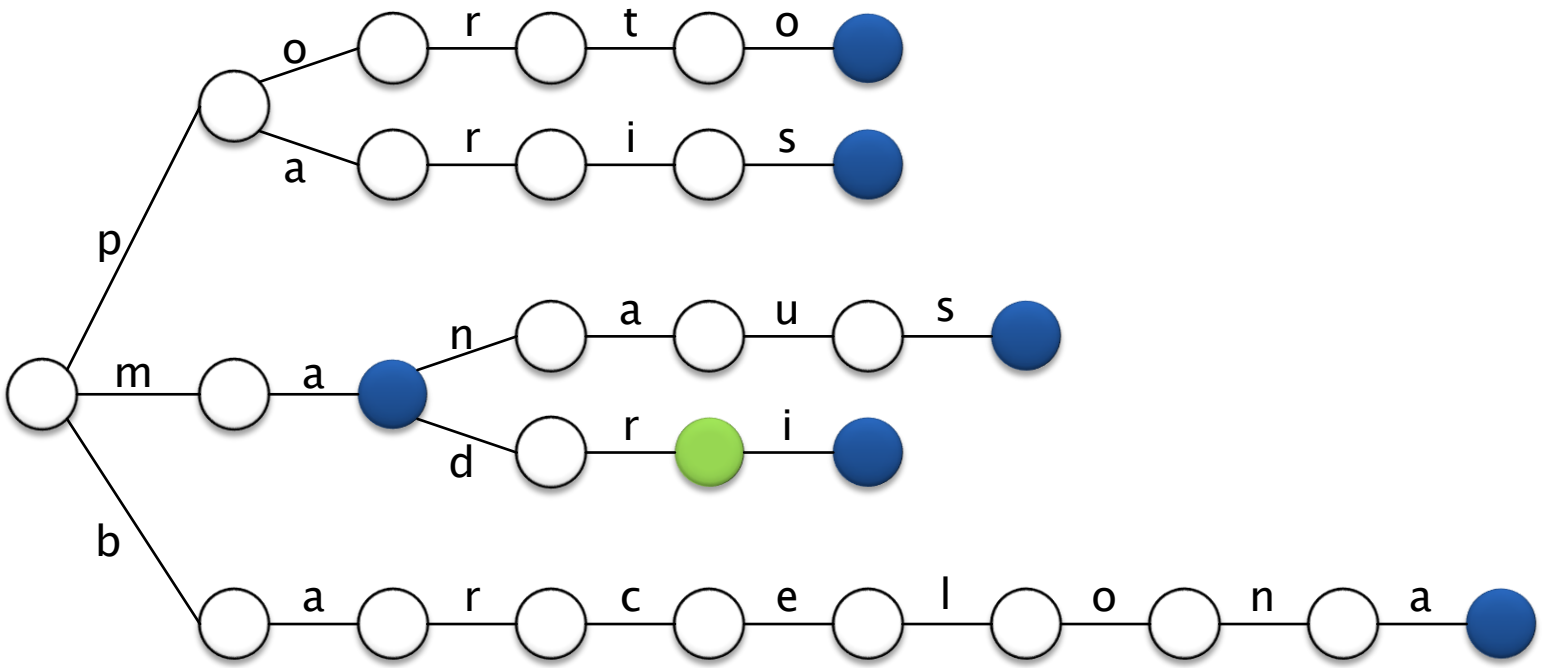
2.3. Remoção

- **Exemplo**: remover a chave **madri**



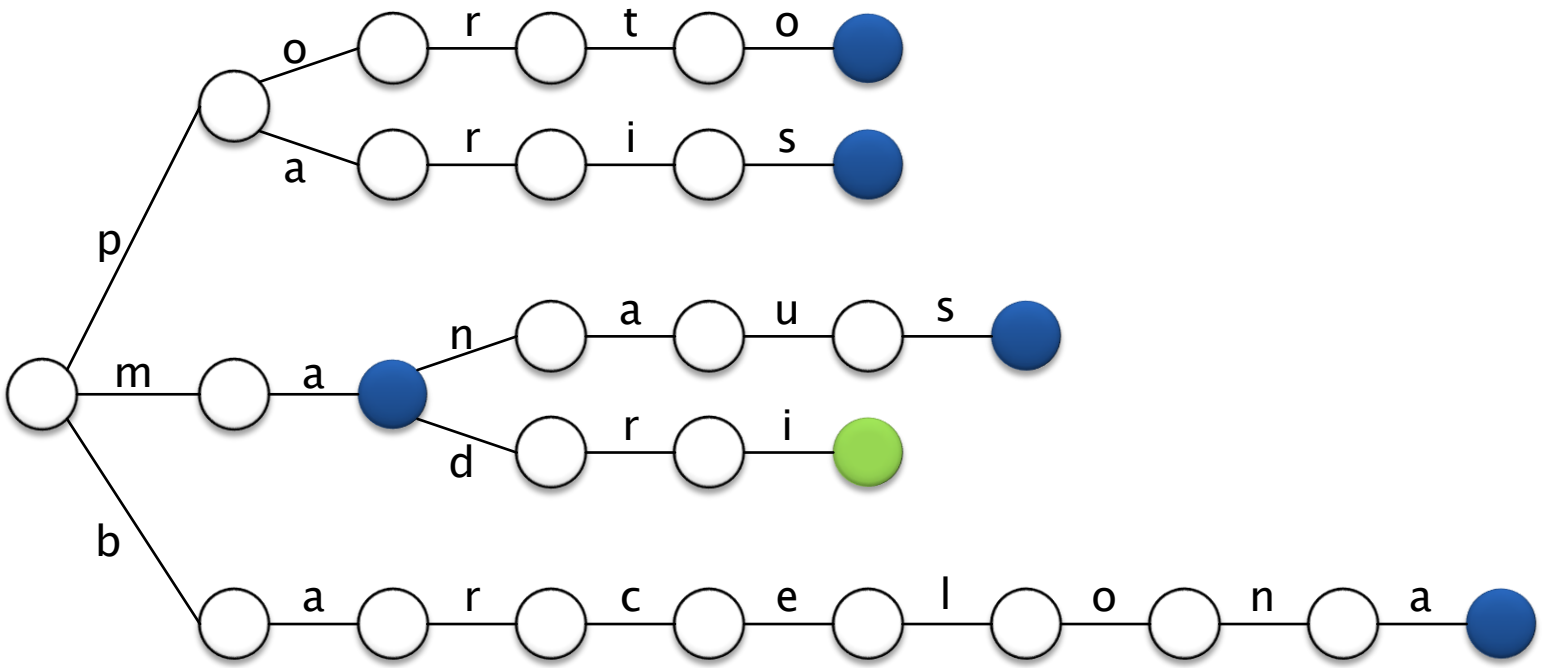
2.3. Remoção

- **Exemplo**: remover a chave **madri**



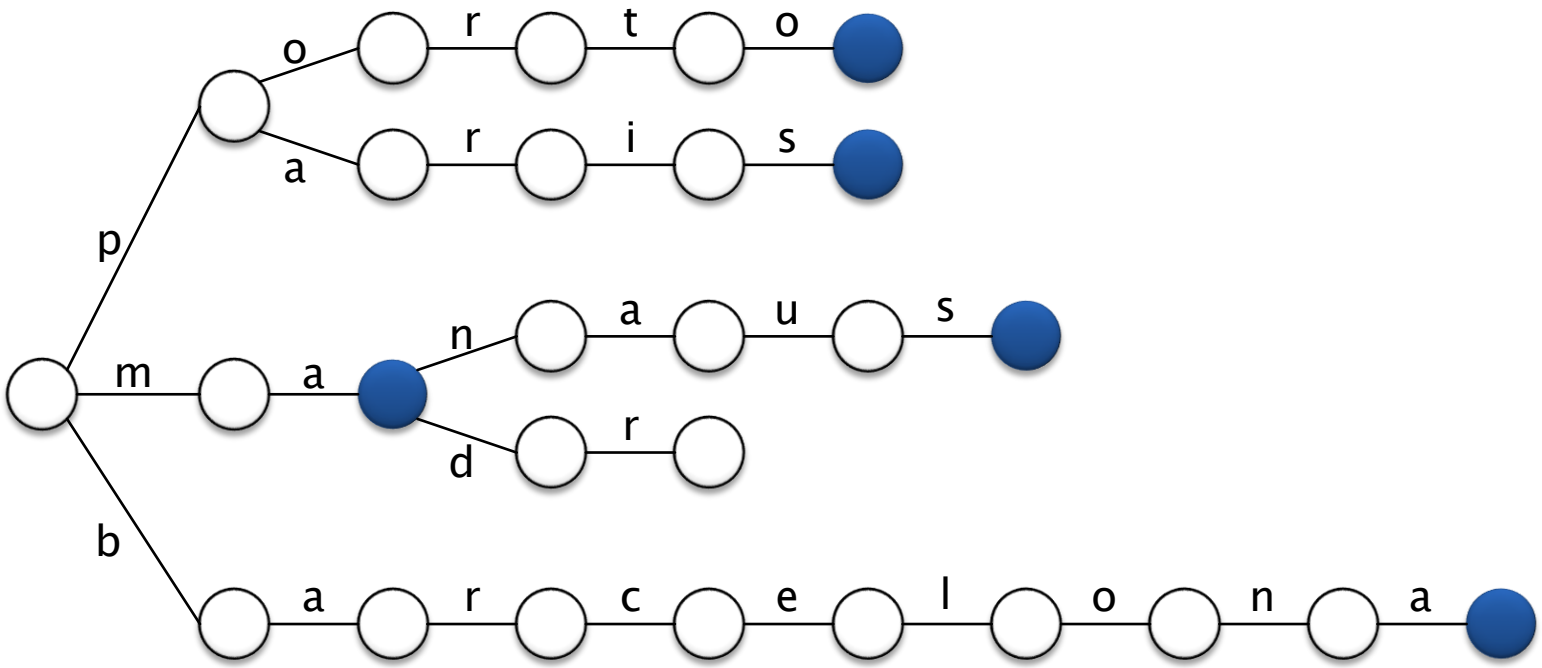
2.3. Remoção

- **Exemplo**: remover a chave **madri**



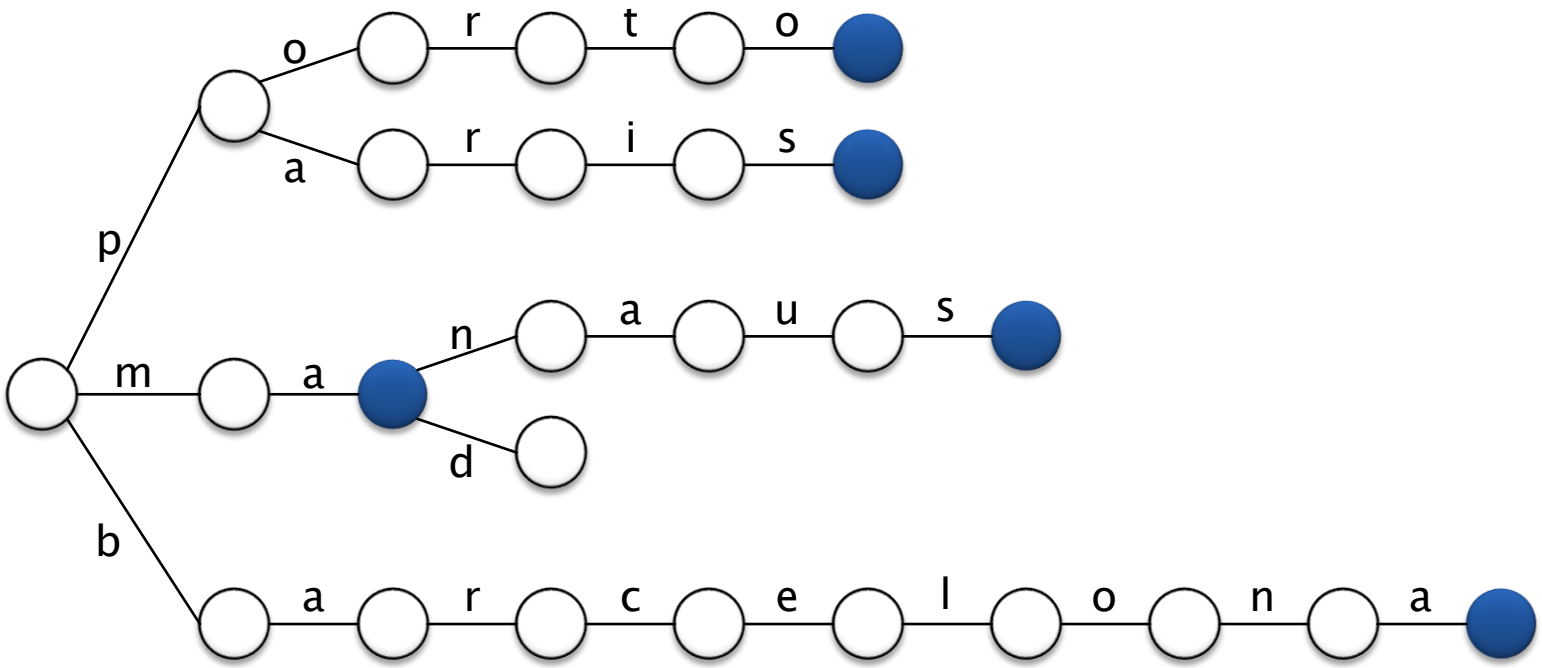
2.3. Remoção

- **Exemplo**: remover a chave **madri**



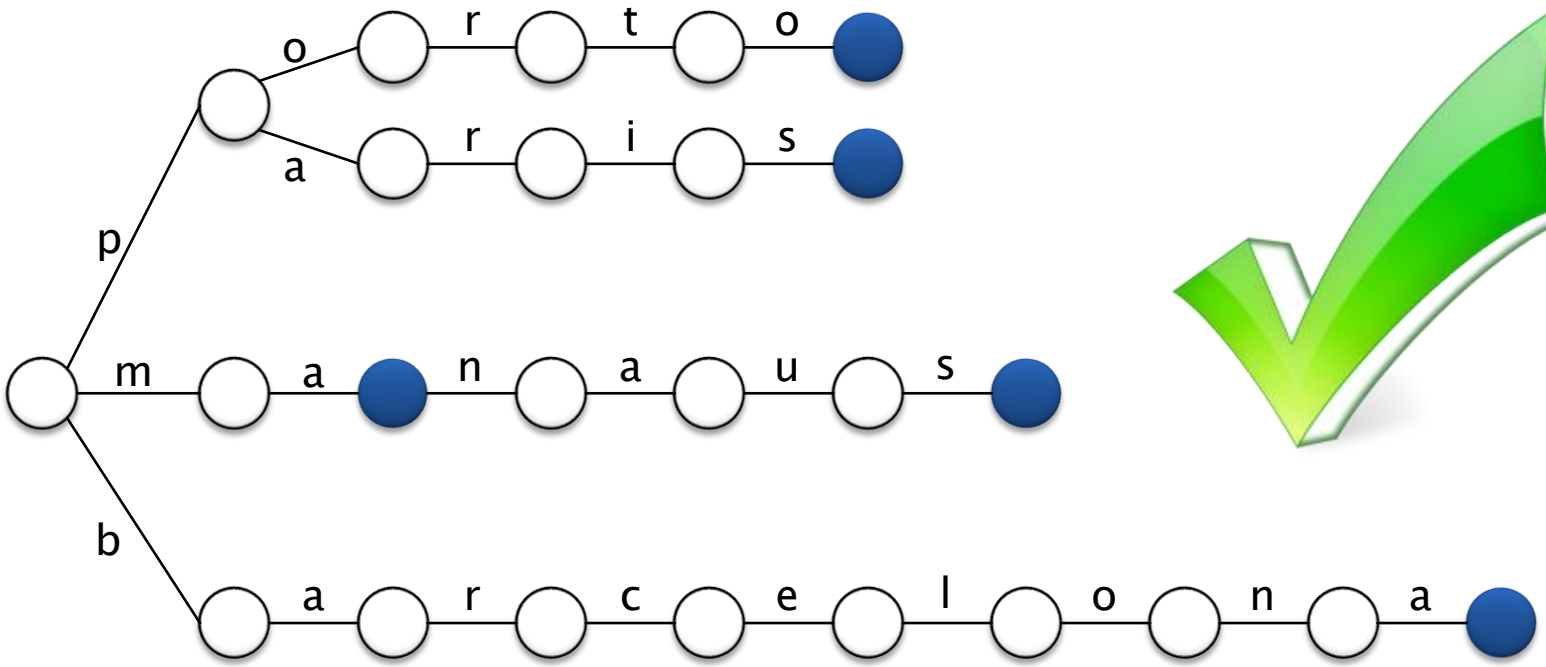
2.3. Remoção

- **Exemplo**: remover a chave **madri**



2.3. Remoção

- **Exemplo**: remover a chave **madri**



2.4. Implementação

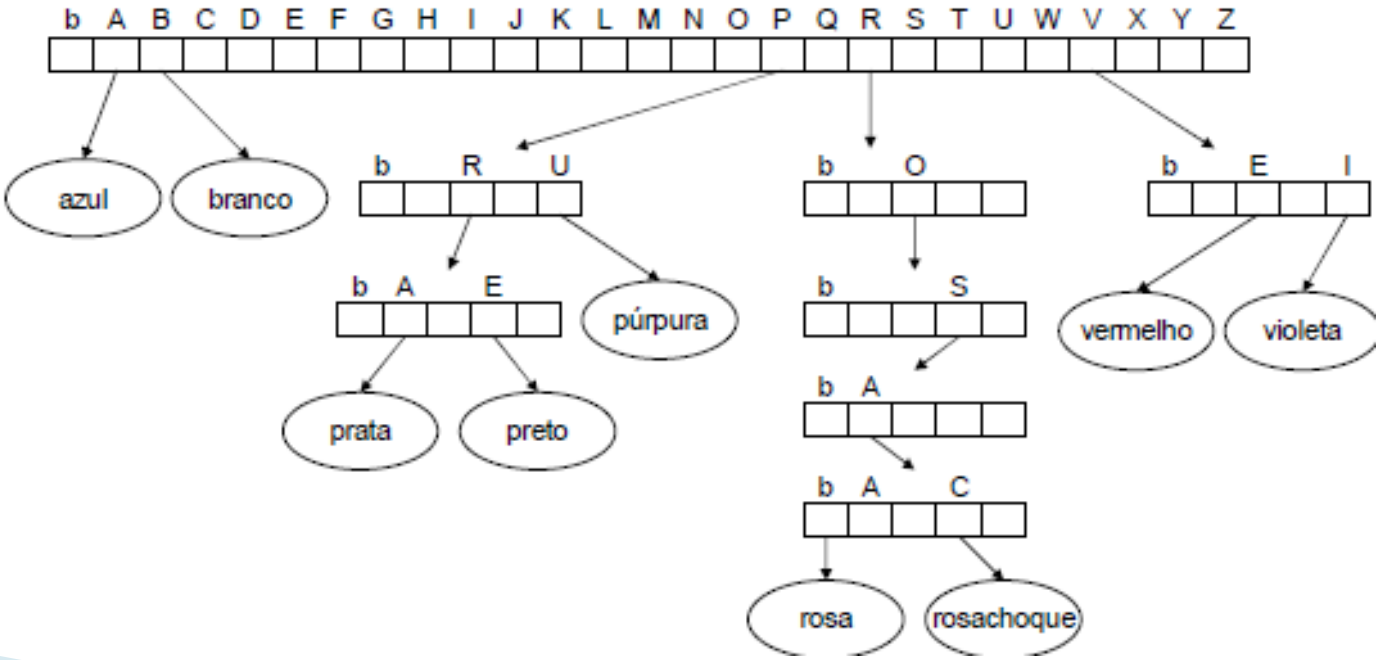
1. Implementação mais simples: **R-WAY**

   - A árvore contém dois tipos de nós:
     - Nó de desvio.
     - Nó de informação.

   - Cada nó de desvio contém <u>todos</u> os valores do alfabeto mais 1 **símbolo especial** para determinar uma **chave.**
     - Há desperdício de espaço.

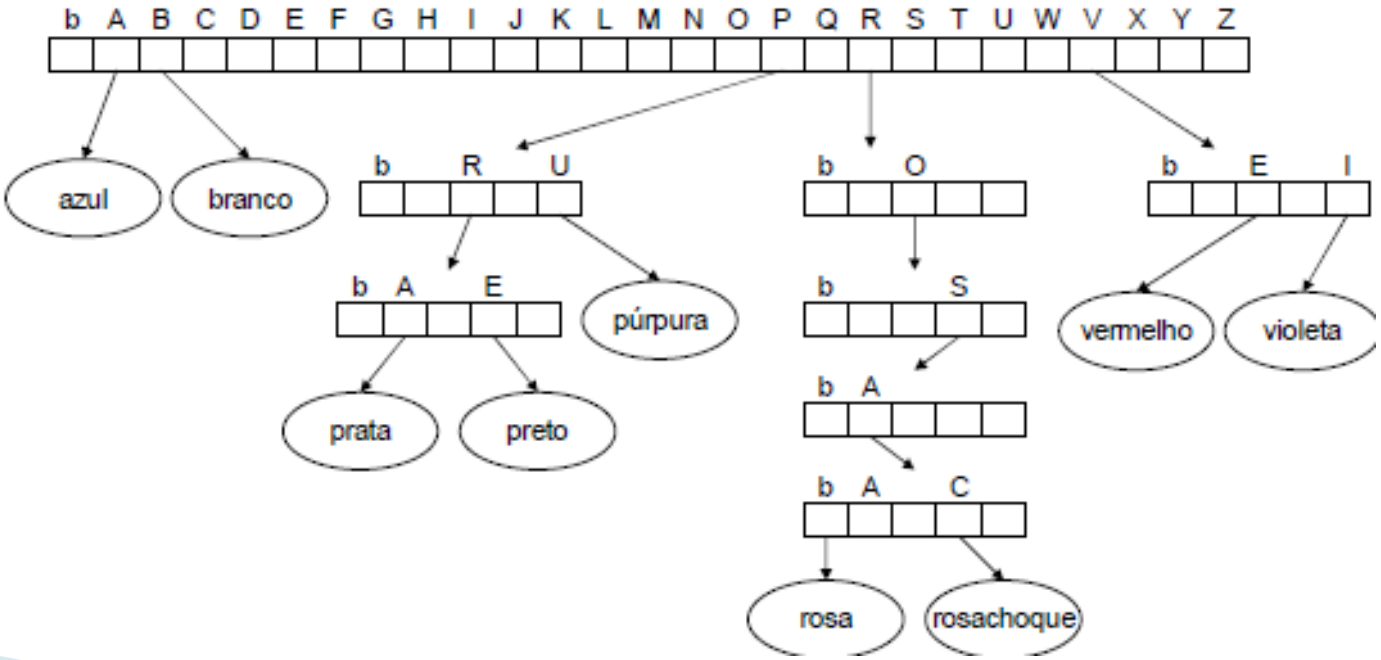
2.4. Implementação

- Considere uma trie para armazenar chaves do alfabeto {a, b, c, d, …, z}.
  - Ou seja, 27 letras.



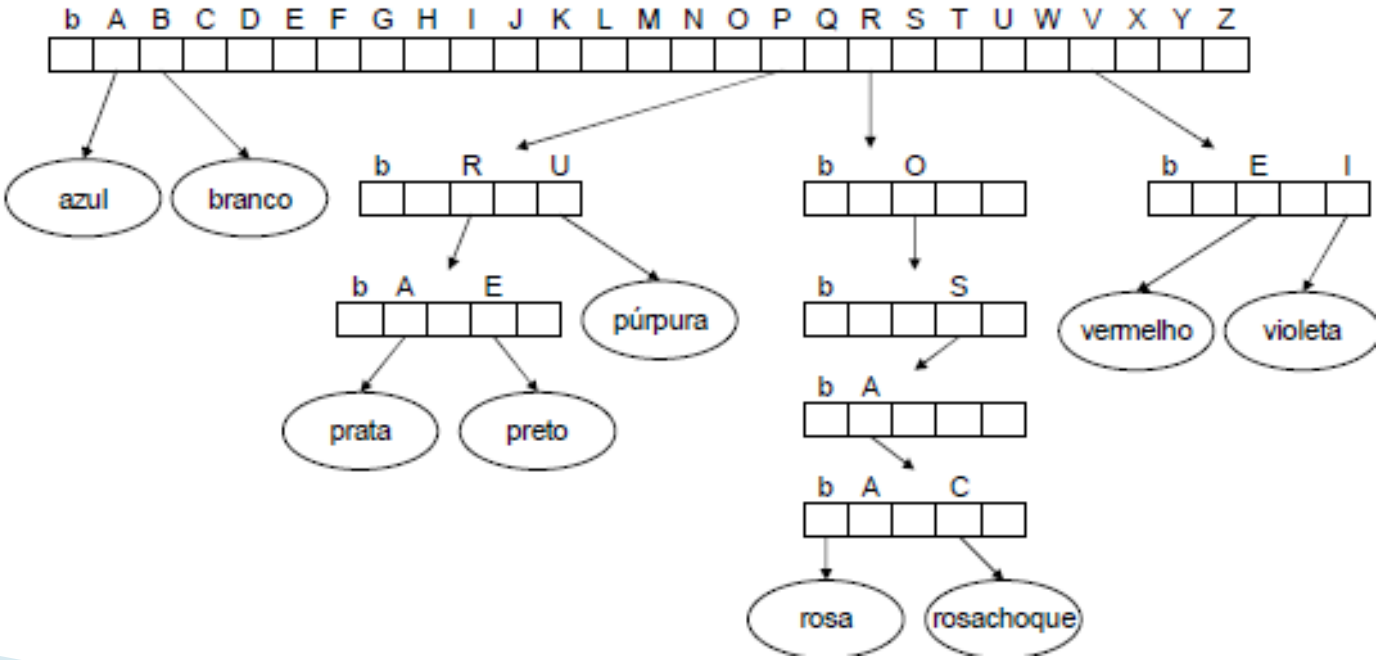
2.4. Implementação

- A árvore seguinte contém dois tipos de nós:
  - Nó de desvio
  - Nó de informação



2.4. Implementação

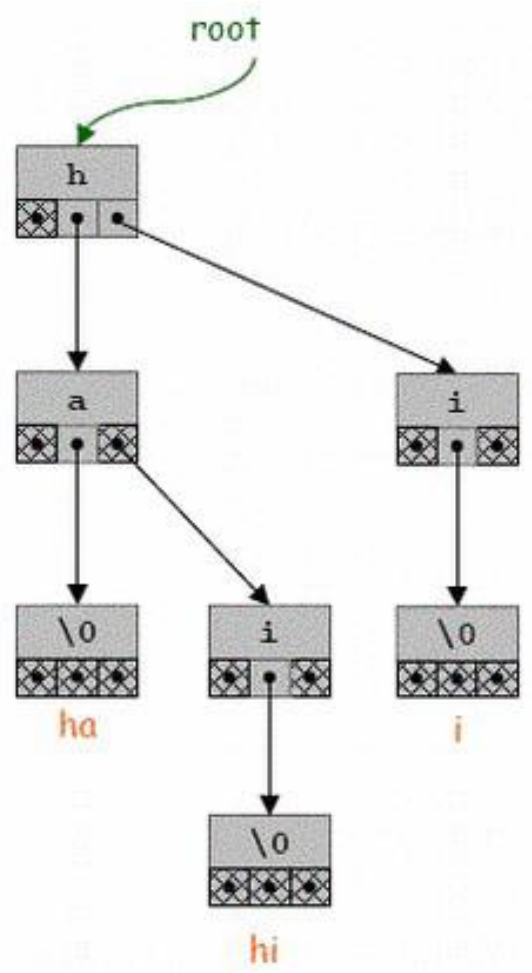
- Nó de desvio contém 27 campos + 1 (b) para determinar uma chave.



2.4. Implementação

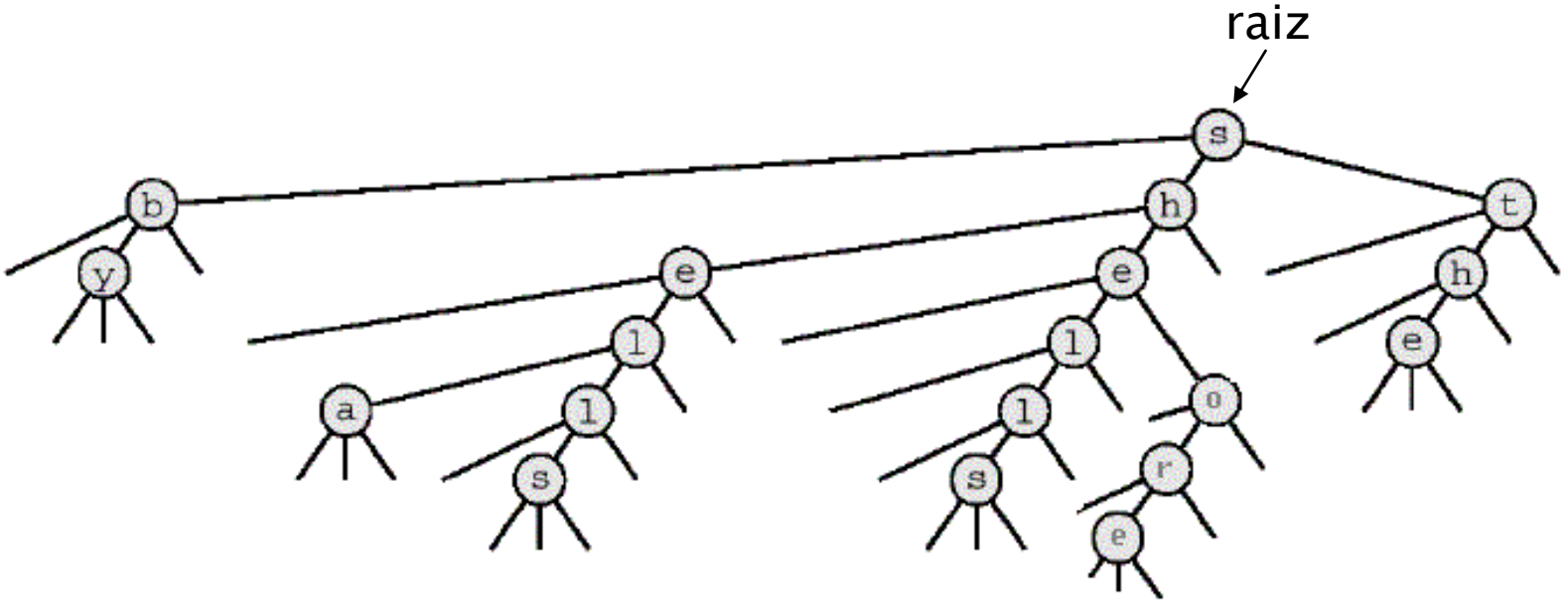
2. **TST – Ternary Search Tree**

   - Árvore de busca composta por três partes.
   - Resolve o problema de desperdício de espaço.
   - Cada nó aloca três ponteiros:
     - **Centro**: caractere seguinte.
     - **Filho da esquerda**: caractere alternativo menor.
     - **Filho da direita**: caractere alternativo maior.
   - Incluir root.



2.4. Implementação

- Para a árvore ficar balanceada as chaves devem estar ordenadas.
- Insere as chaves a partir da chave central (semelhante a busca binária).



Chaves: by, sea, sells, shells, shore, the

2.5. Análise

- A altura da árvore é igual ao comprimento da chave mais longa.
- O tempo de execução das operações não depende do número de elementos da árvore.
- A utilização de uma trie só compensa se o acesso aos componentes individuais das chaves for bastante rápido.
- Quanto maior a estrutura mais eficiente o uso do espaço.
- Para enfrentar o desperdício de espaço com estruturas pequenas foram criadas as **Árvores Digitais Binárias** e a **Árvore Patricia**.

3. Árvores Digitais Binárias

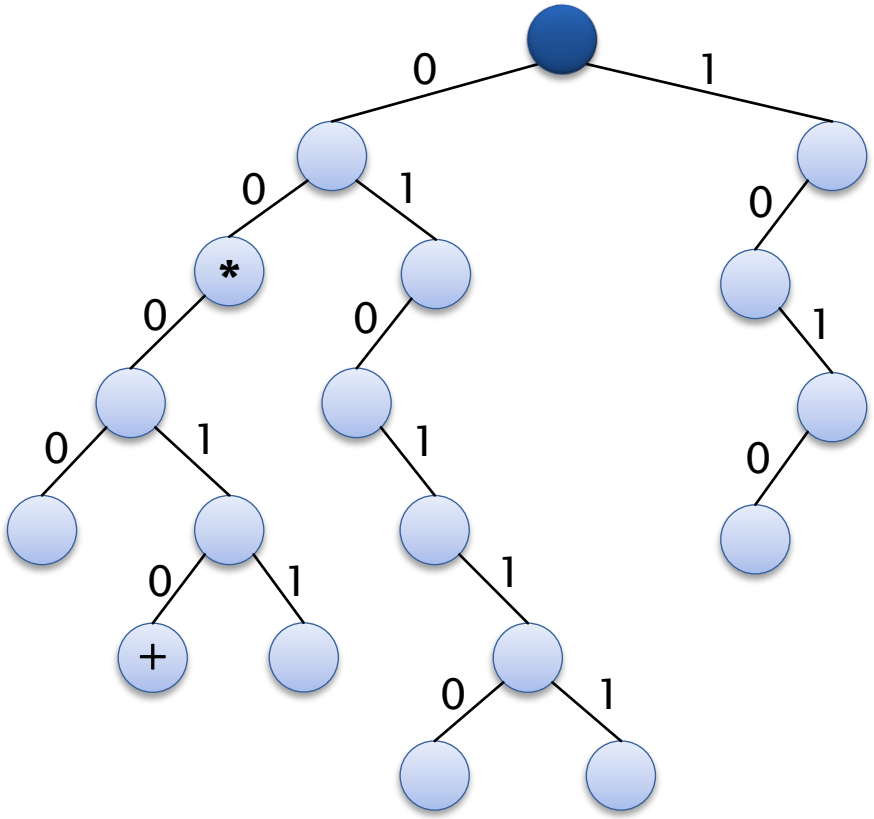
- Árvore digital binária é simplesmente o caso binário da trie, ou seja, uma árvore **m-ária** com m=2.
  - Neste caso, representa-se o alfabeto por {0,1}.
- A seleção do filho **esquerdo** de um nó é interpretada como o **dígito 0** e o **direito** como **1**.
- A maior utilização de árvores digitais dá-se, possivelmente, nesse caso binário.
  - Chaves ou códigos binários são os mais empregados na computação.

3. Árvores Digitais Binárias

- Exemplo:

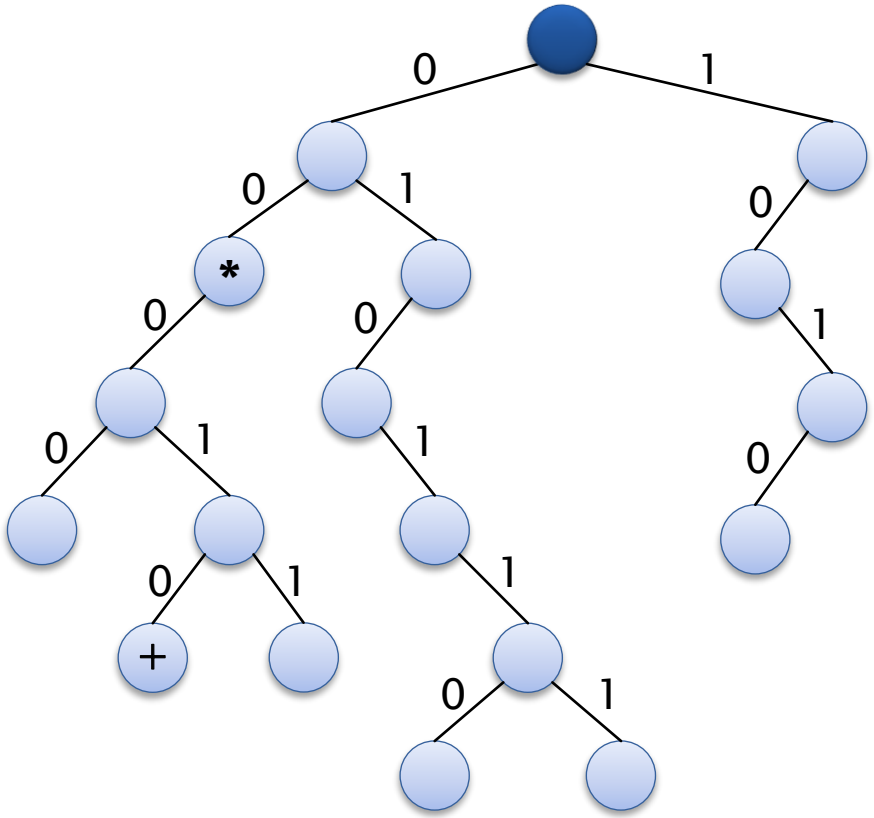
Chaves

```
00  ⇨ *
0000
00010 ⇨ +
00011
010110
010111
10
101
1010
```



3. Árvores Digitais Binárias

- Caso sejam gravadas somente as chaves `{1010, 010110, 010111, 0000, 00010, 00011}`:
  - Zig-zags desnecessários.
  - Maior espaço de memória ocupado desnecessariamente.
- Alternativa
  - Criar a árvore tentando **reduzir** os zig-zags inúteis.



3.1. Árvores Binárias de Prefixos

- Ao analisar as chaves, verificamos que algumas são prefixos de outras na coleção.
  - Por exemplo:

00	**010110**	**10**
0000	**010111**	**101**
00010		**1010**
00011		

- Isso corresponde a dizer que o caminho da raiz até o nó de chave **00** é parte do caminho da raiz até o nó de chave **00010**.
- Frequentemente, para melhor manipular a estrutura, deseja-se que tal situação não aconteça.
  - Assim, uma árvore binária de prefixo é uma árvore digital binária tal que **nenhum código seja prefixo do outro**.

3.1. Árvores Binárias de Prefixos

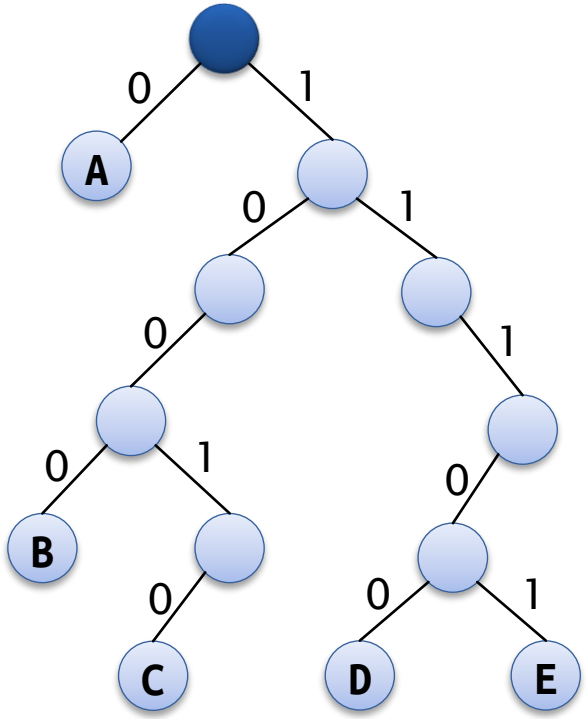
A = 0

B = 1000

C = 10010

D = 11100

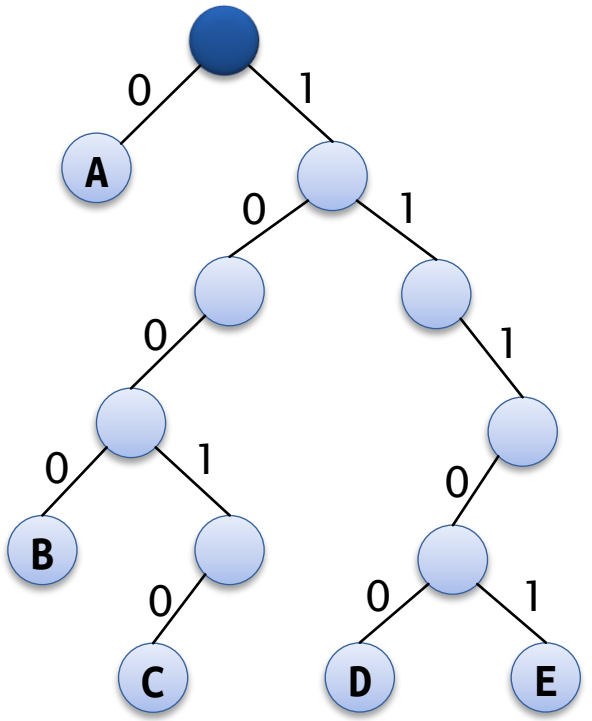
E = 11101



3.1. Árvores Binárias de Prefixos

- Uma propriedade interessante da árvore binária de prefixo é que há uma correspondência entre o **conjunto de chaves** e o das **folhas das árvores**.

- Isto é, cada chave é unicamente representa por uma folha e a codificação binária dessa chave corresponde ao caminho da raiz até essa folha.



3.1. Árvores Binárias de Prefixos

- É possível simplificar a estrutura das Árvores Binárias de Prefixo evitando adicionar todo o caminho até o nó, caso o caminho até o nó seja o único a ser percorrido.

- Como implementar essa abordagem?
  - **Árvore Patricia**

4. Árvore Patricia

- Patricia é abreviatura de:
  - ***P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric***
  - Algoritmo Prático para Recuperar Informação Codificada em Alfanumérico.
- O algoritmo para construção da árvore Patricia é baseado no método de pesquisa digital, mas sem apresentar o inconveniente das tries.
  - É construída a partir da árvore binária de prefixo.

4. Árvore Patricia

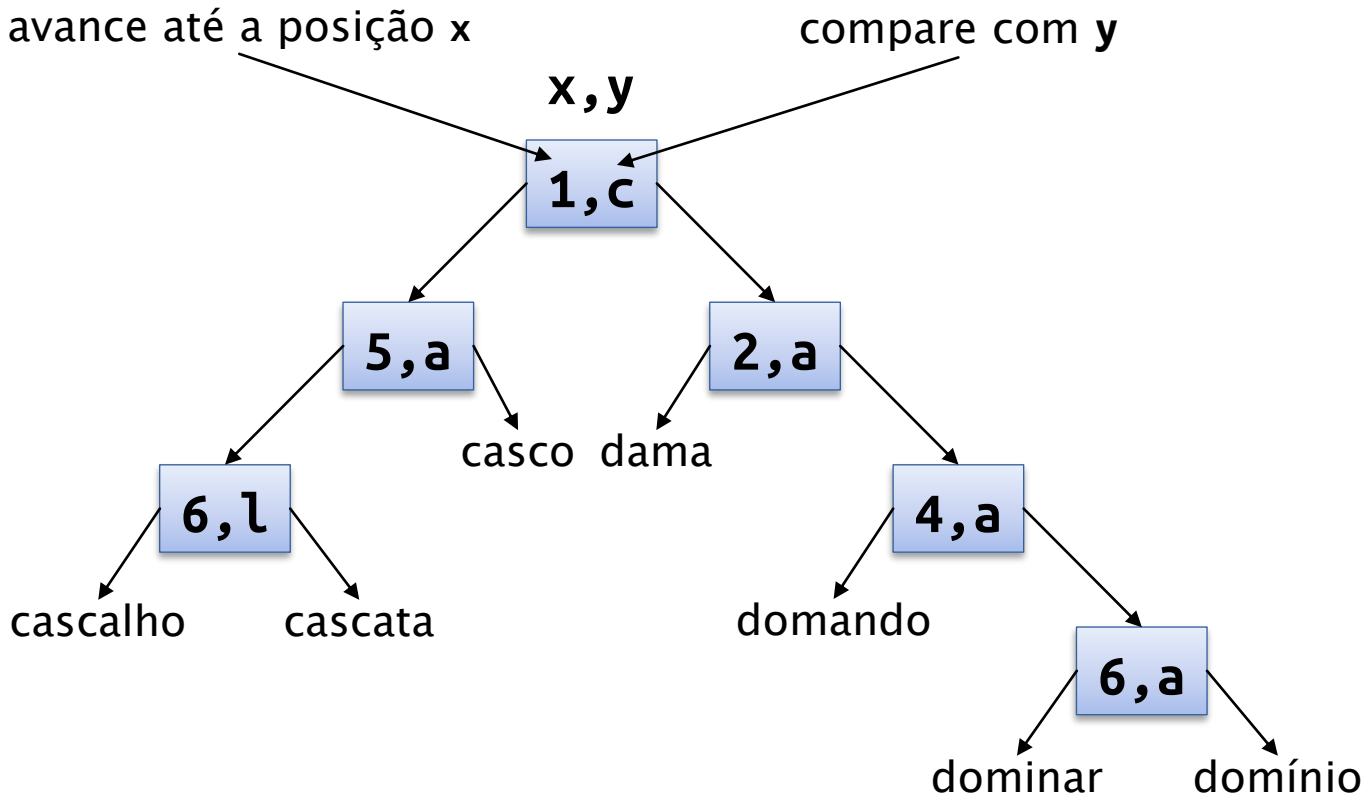
- O problema de caminhos de uma só direção é eliminado por meio de uma solução simples e elegante:
  - Cada nó interno da árvore contém o índice do caractere a ser testado para decidir qual sub-árvore seguir.
- Em outras palavras, caminhos que possuem nós com **apenas 1 filho** são **agrupados** em uma única aresta.

4. Árvore Patricia

- Os NÓS contém:
  - **y**: caractere que deve ser comparado.
  - **x**: índice da posição na chave onde se deve efetuar a comparação.
- Se menor ou igual a **y** avança a esquerda, se maior que **y** avança a direita.
- As chaves válidas encontram-se nas folhas da árvore.

4. Árvore Patricia

- Exemplo de Representação:

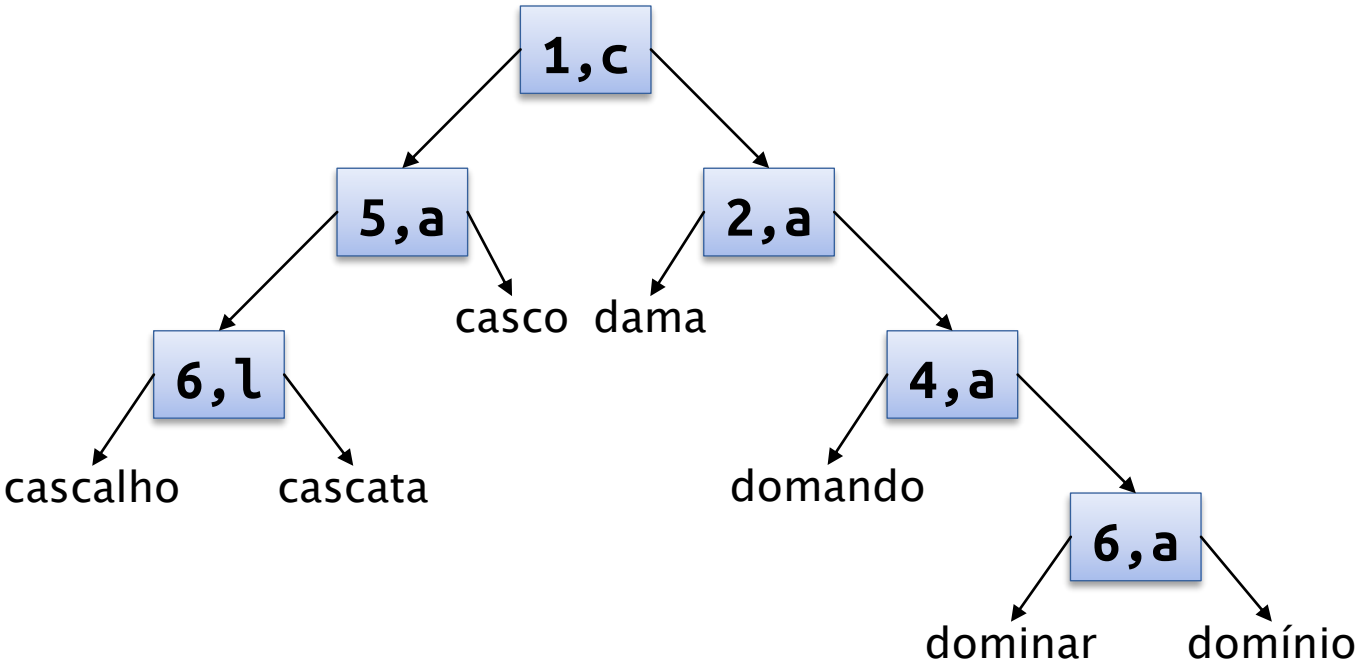


4.1. Busca

- **Sequencia de passos:**
  - Comparar o caractere na posição **x** com o caractere **y**.
    1. Se menor ou igual: Segue ramo a esquerda
    2. Se maior: Segue ramo a direita.
  - Repete isso até chegar numa folha.

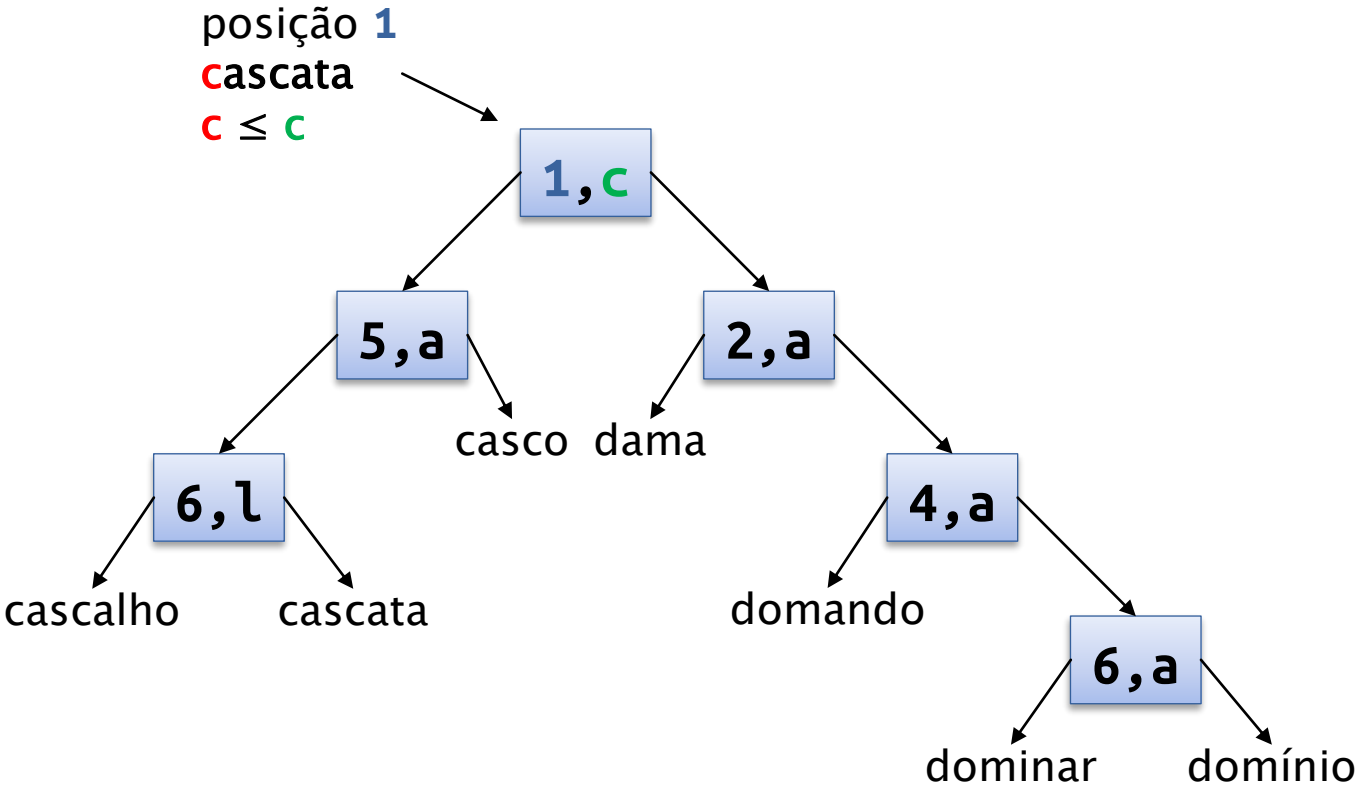
4.1. Busca

- Exemplo 1: buscar por **cascata**



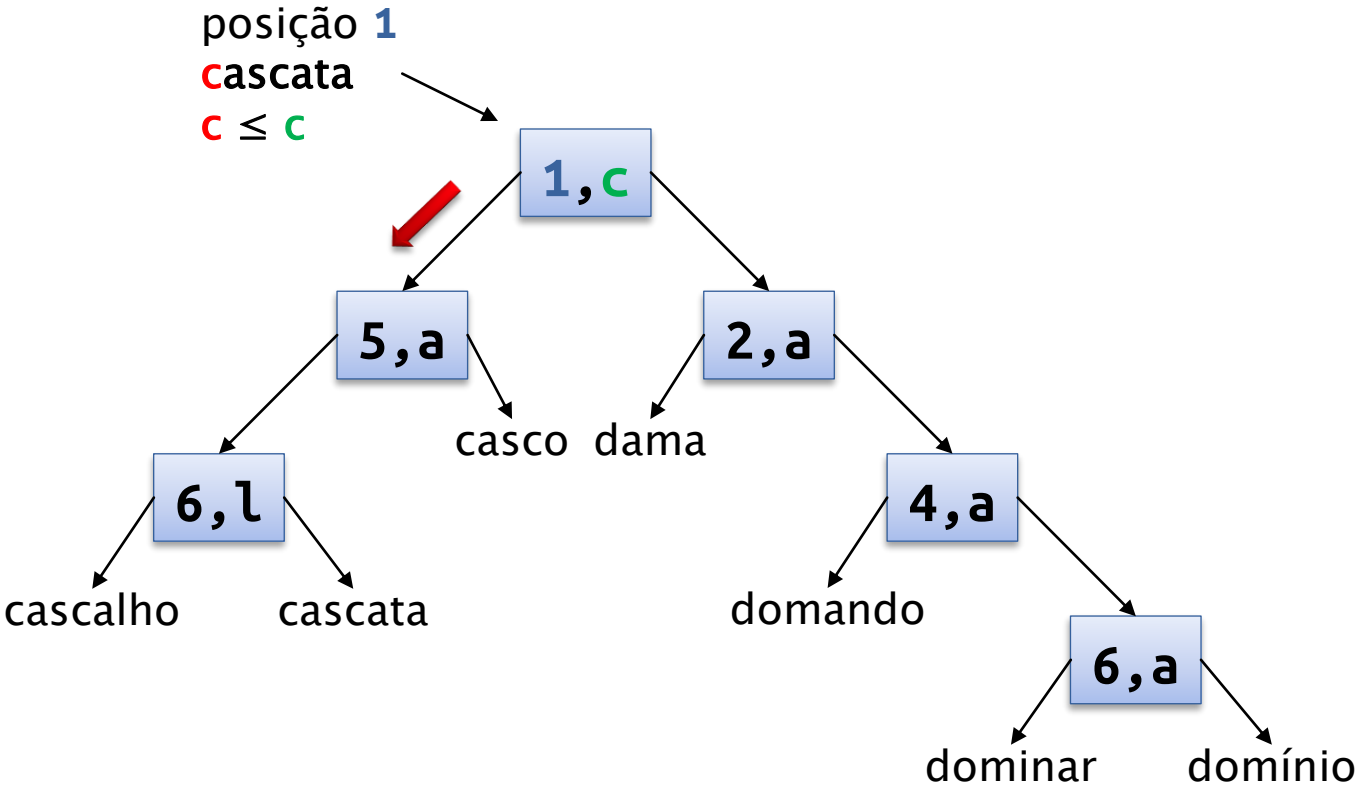
4.1. Busca

- Exemplo 1: buscar por **cascata**



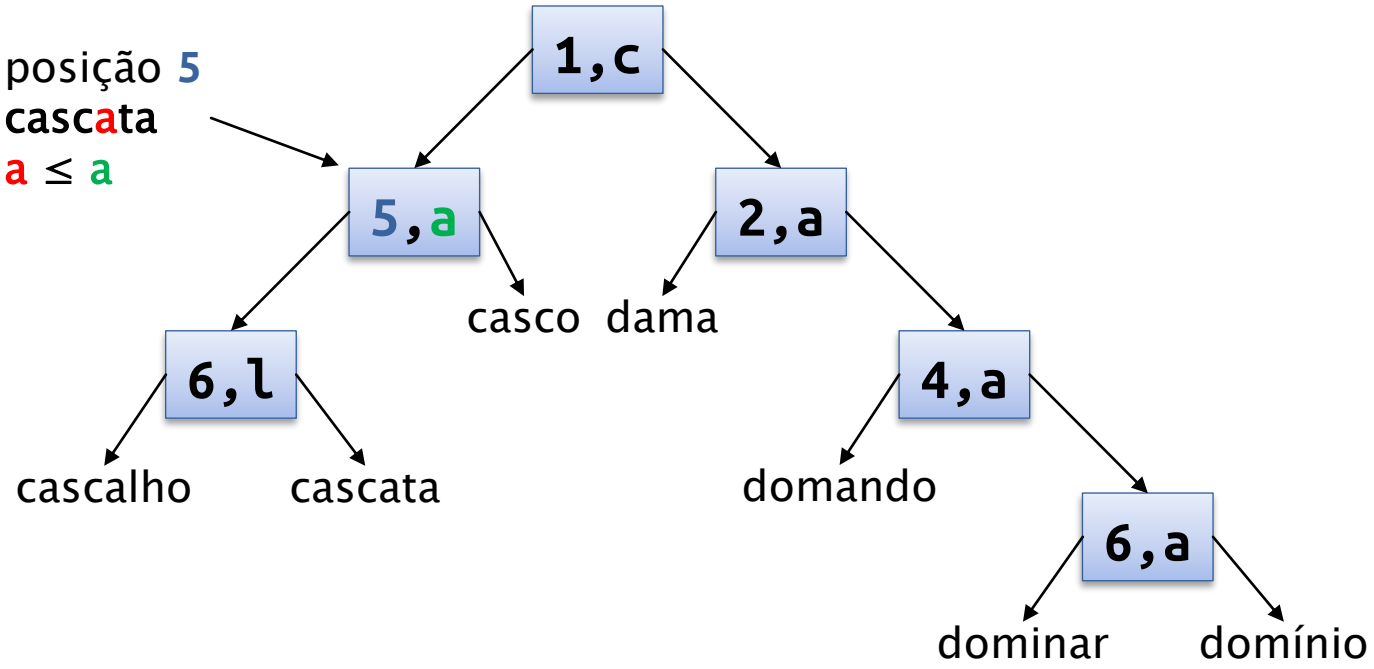
4.1. Busca

- Exemplo 1: buscar por **cascata**



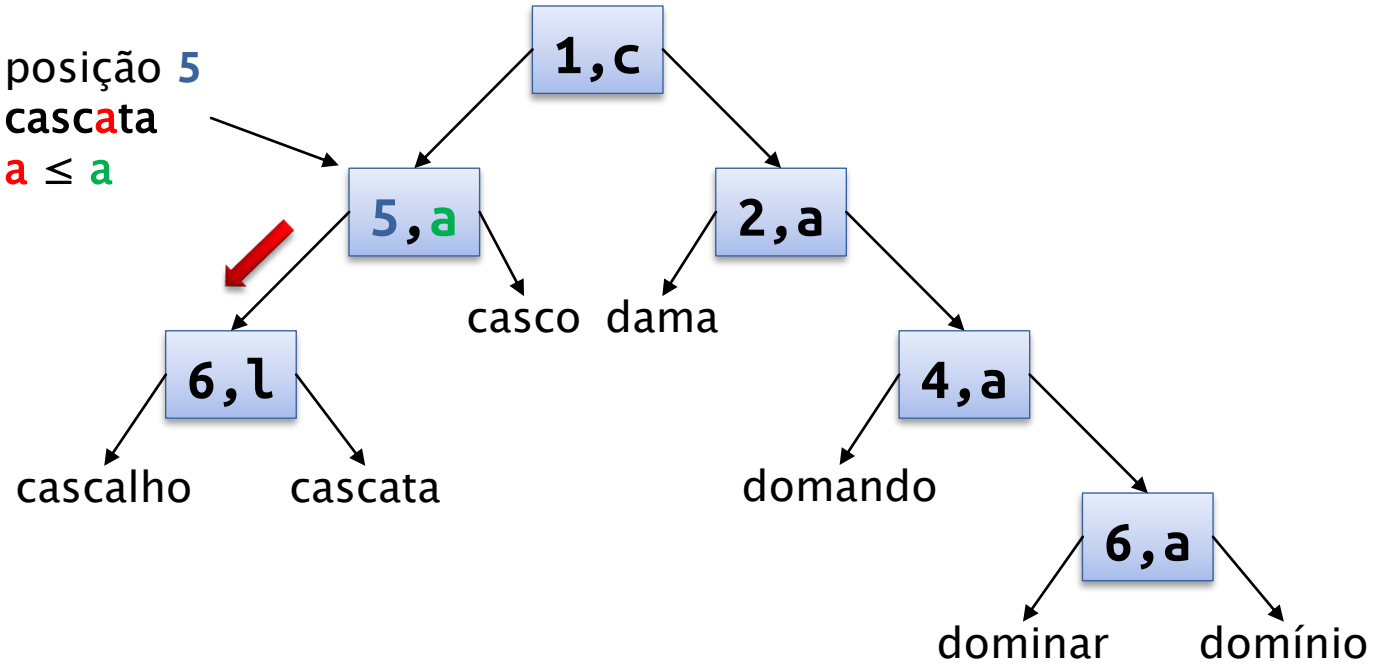
4.1. Busca

- Exemplo 1: buscar por **cascata**



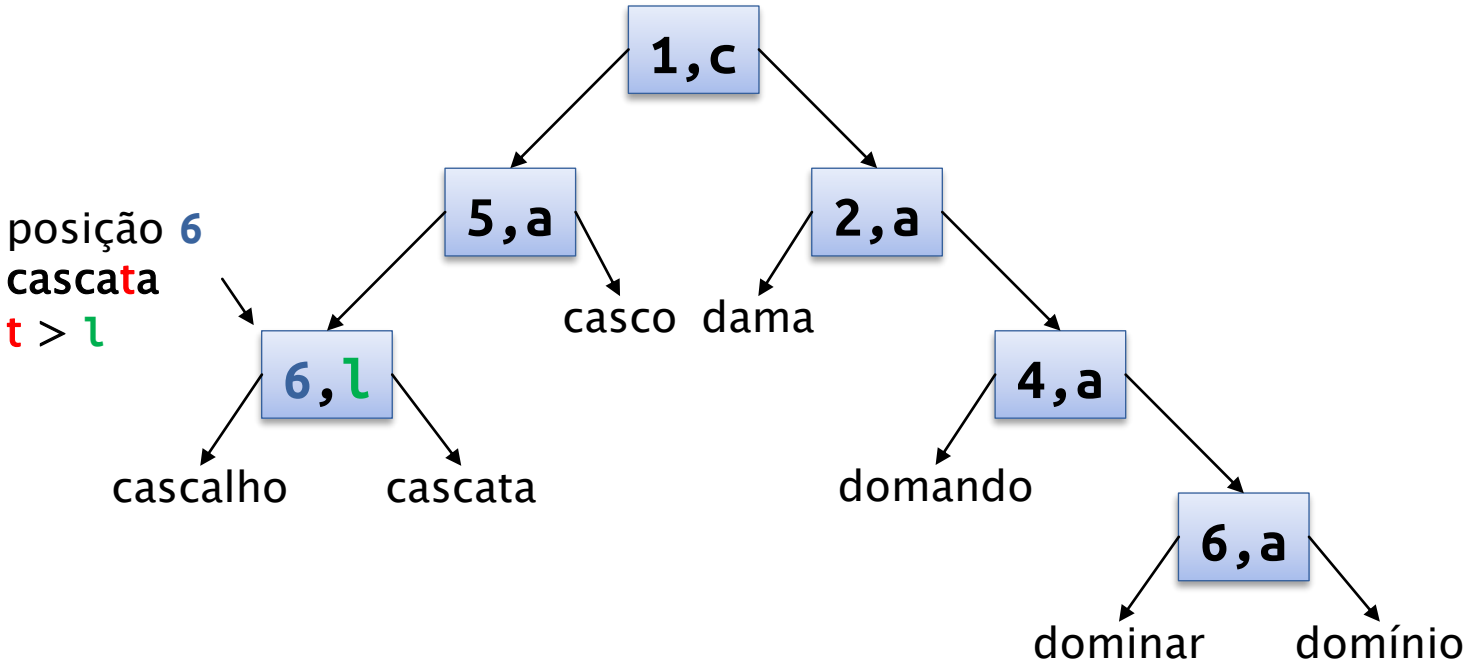
4.1. Busca

- Exemplo 1: buscar por **cascata**



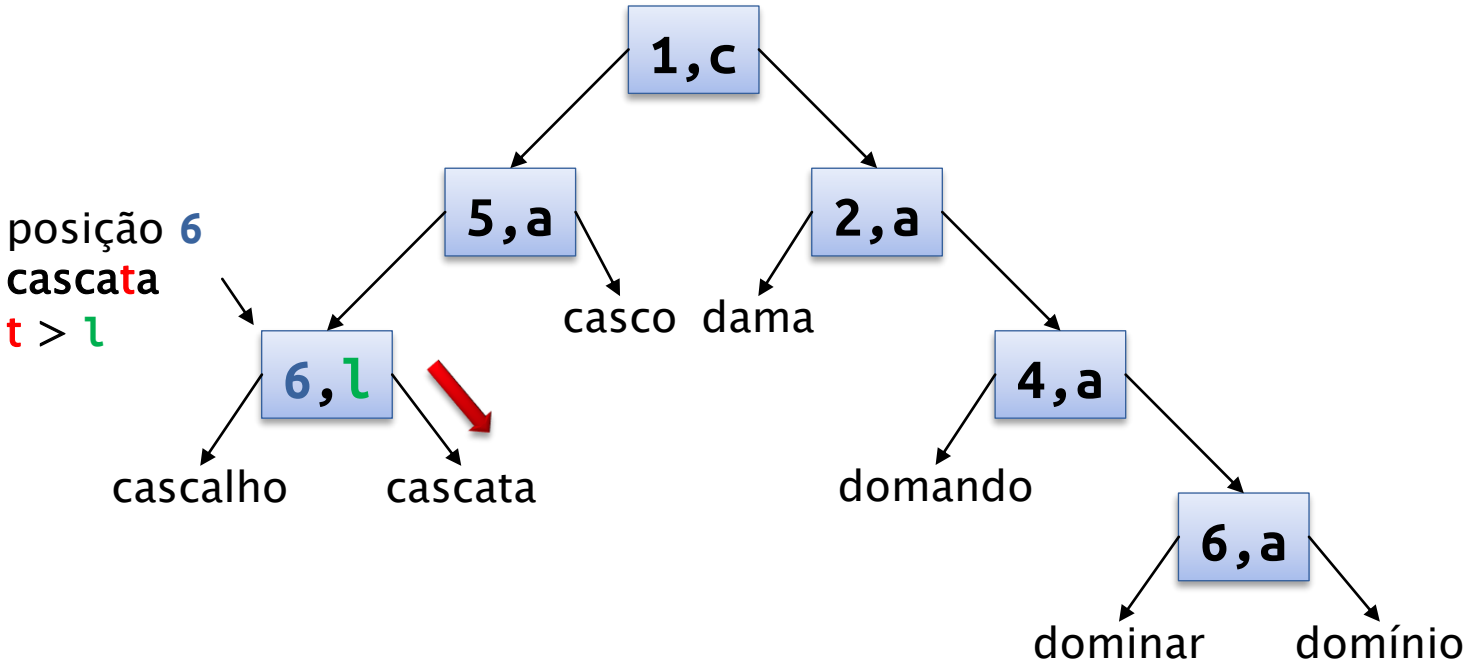
4.1. Busca

- Exemplo 1: buscar por **cascata**



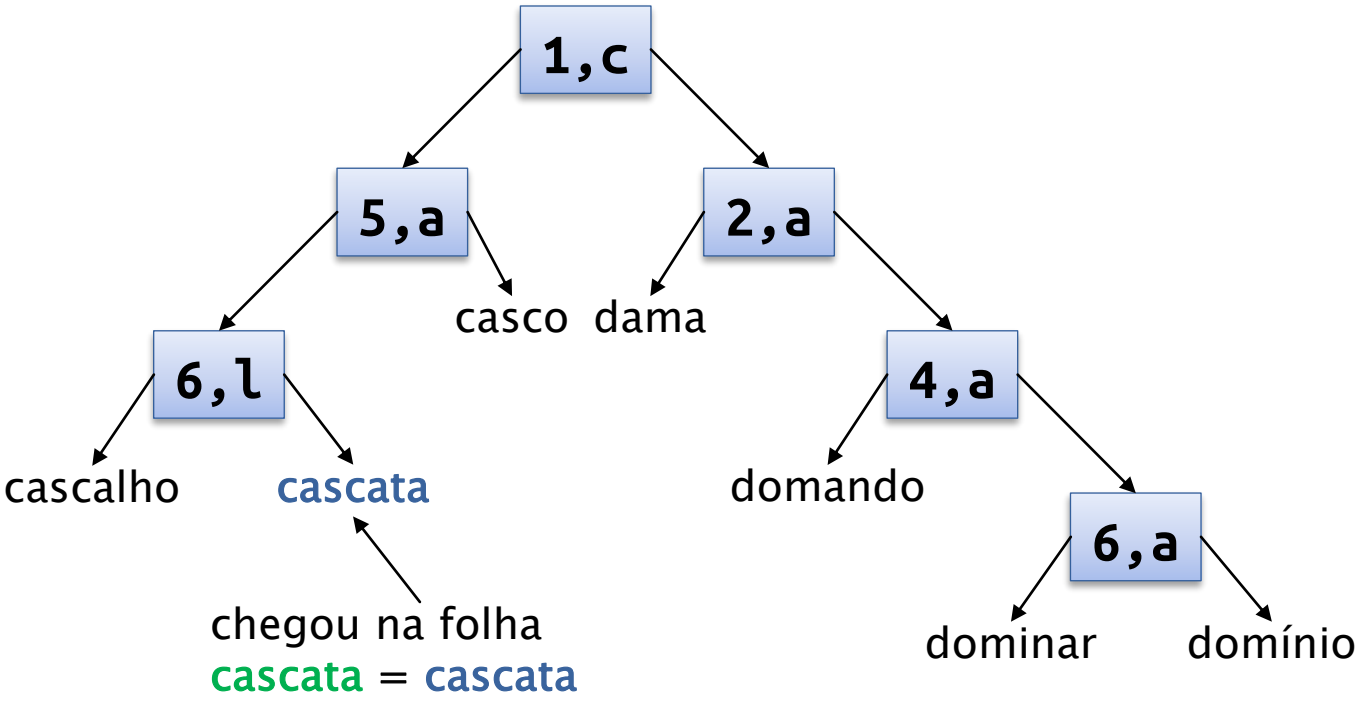
4.1. Busca

- Exemplo 1: buscar por **cascata**



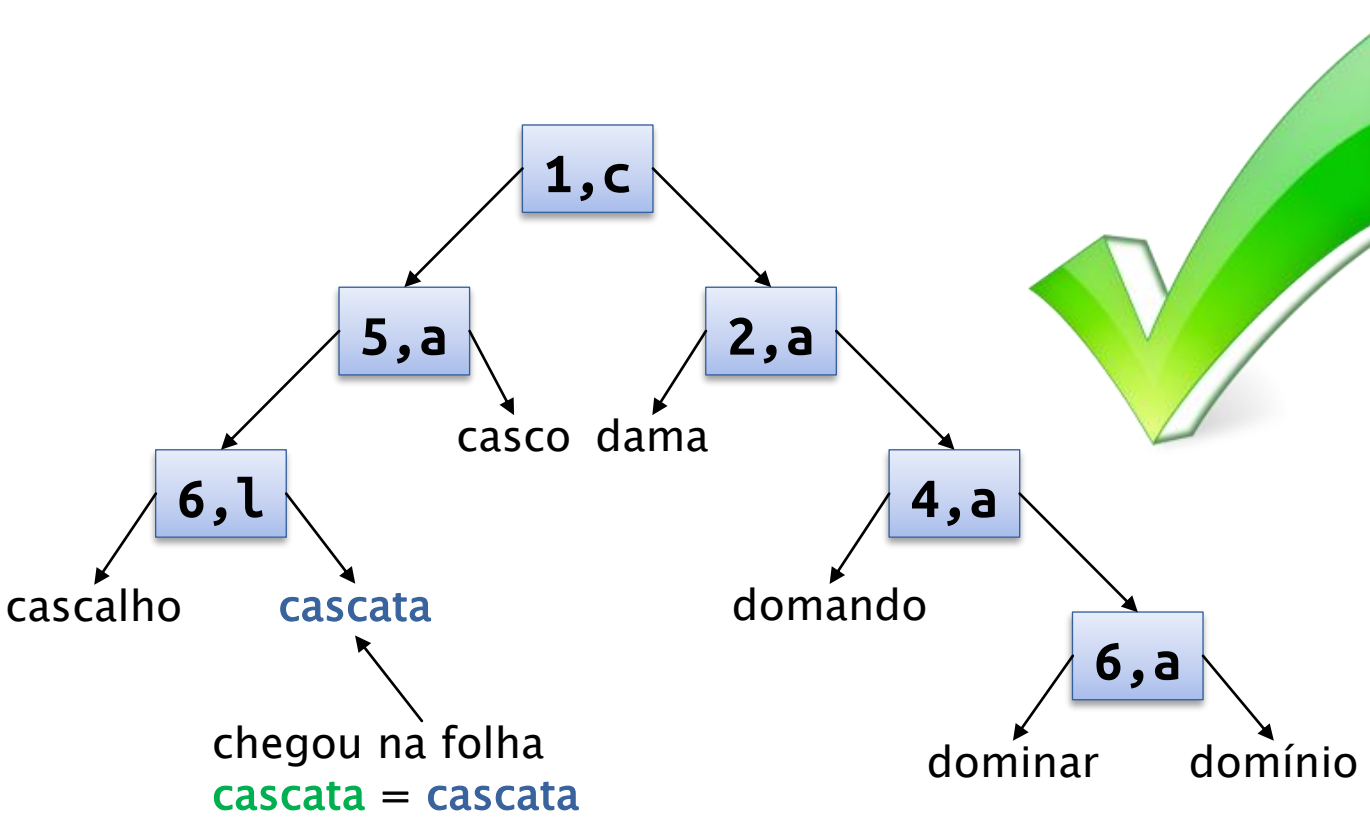
4.1. Busca

- Exemplo 1: buscar por **cascata**



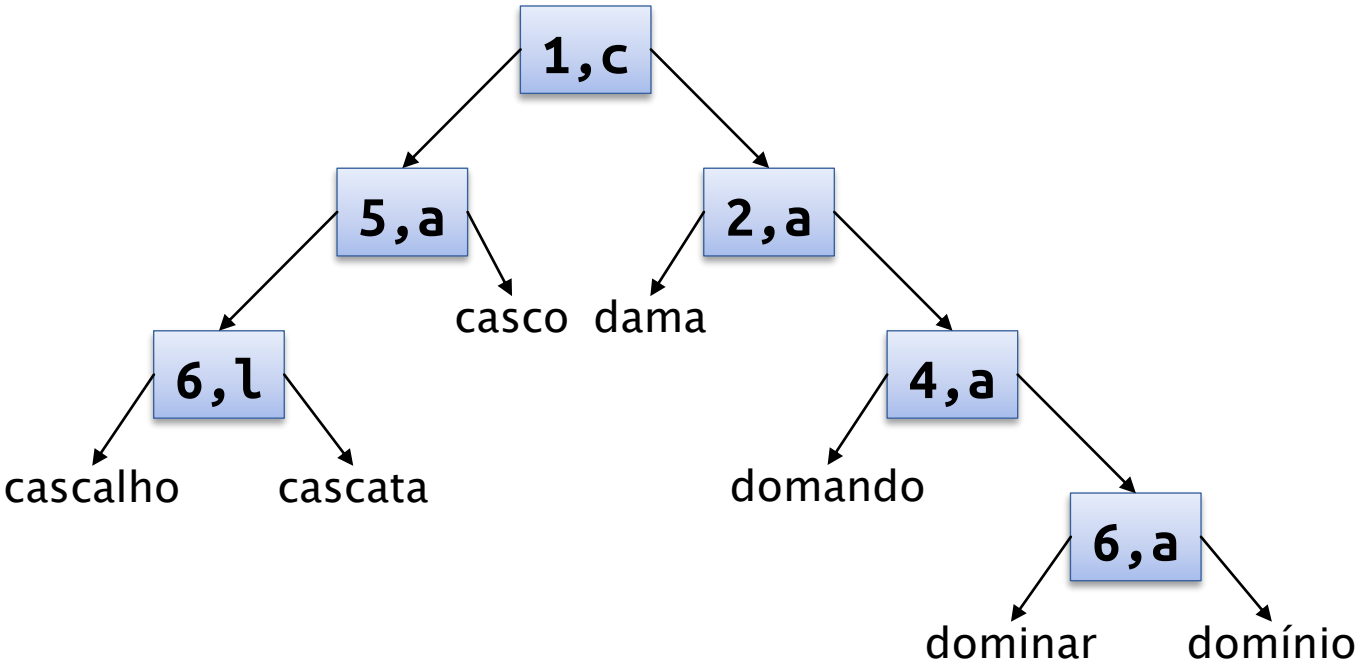
4.1. Busca

- Exemplo 1: buscar por **cascata**



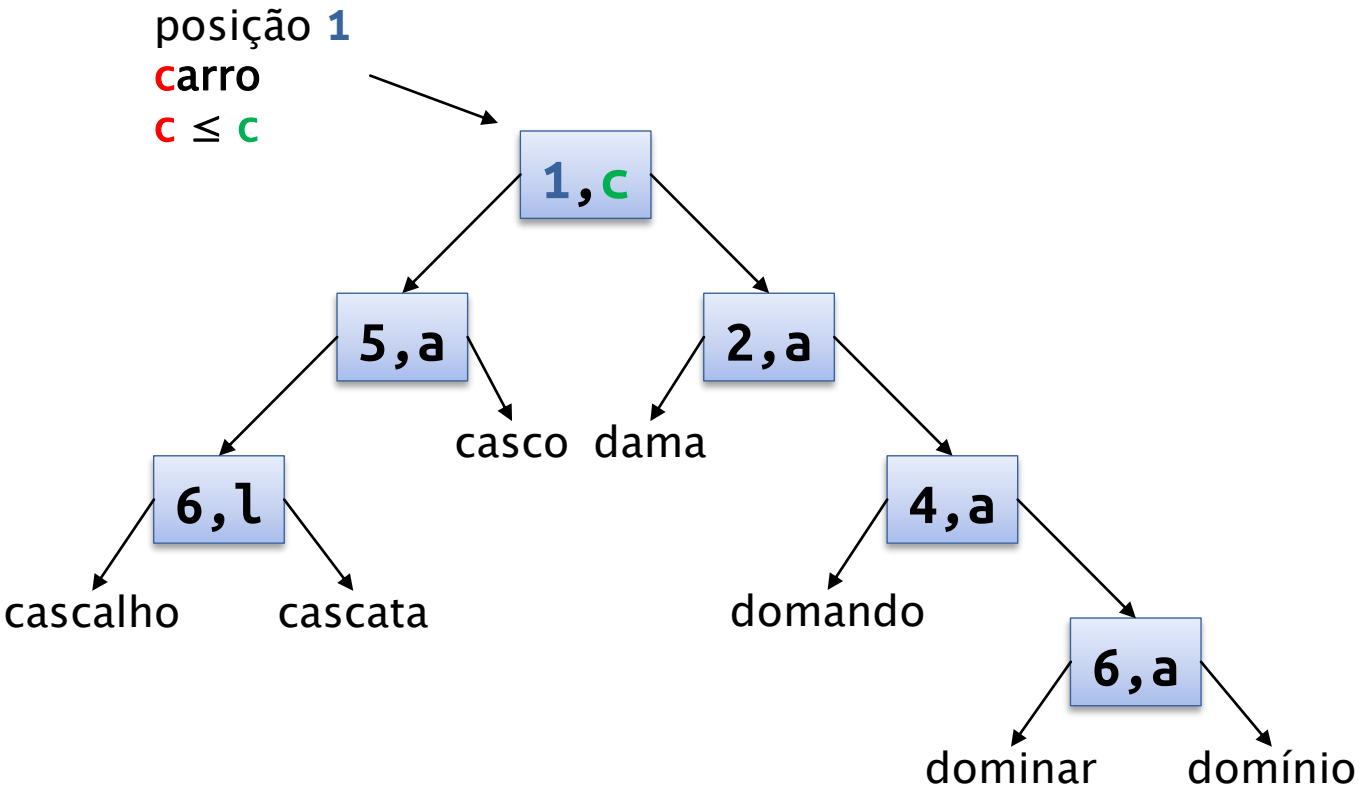
4.1. Busca

- Exemplo 2: buscar por **carro**



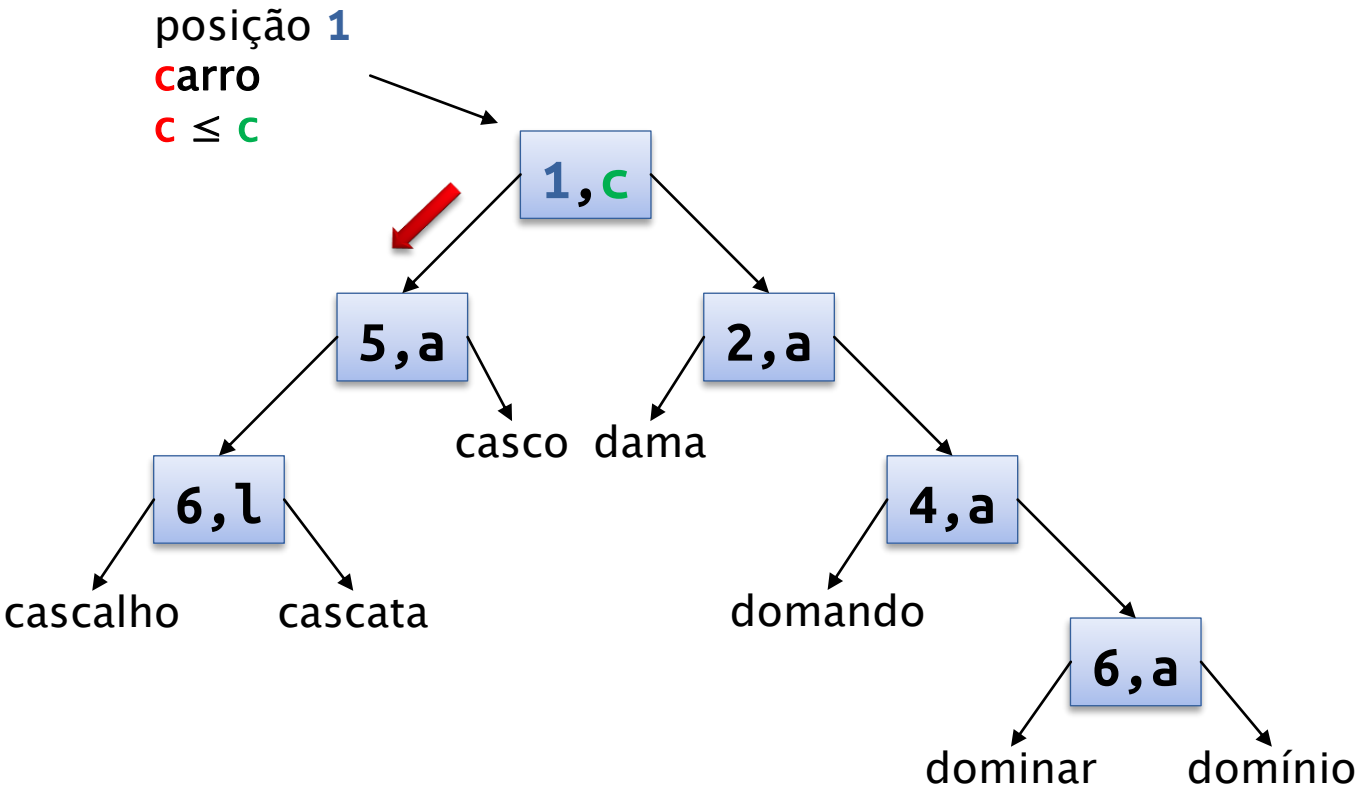
4.1. Busca

- Exemplo 2: buscar por **carro**



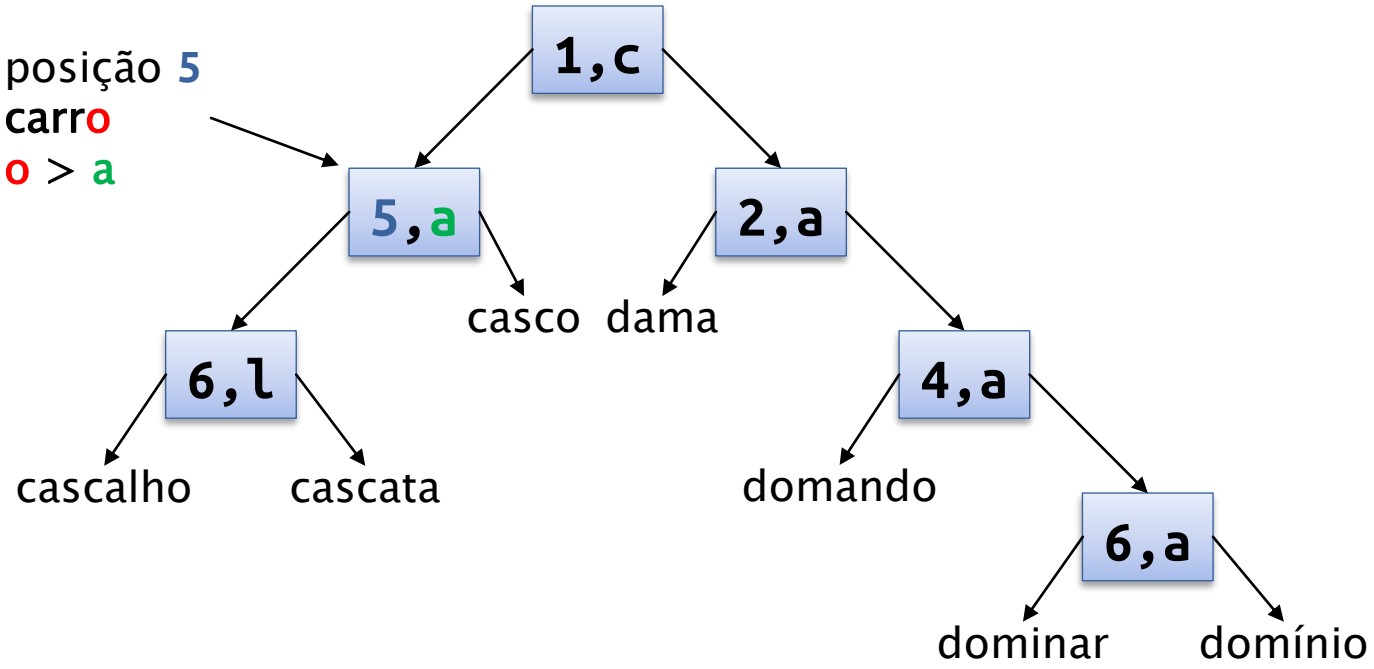
4.1. Busca

- Exemplo 2: buscar por **carro**



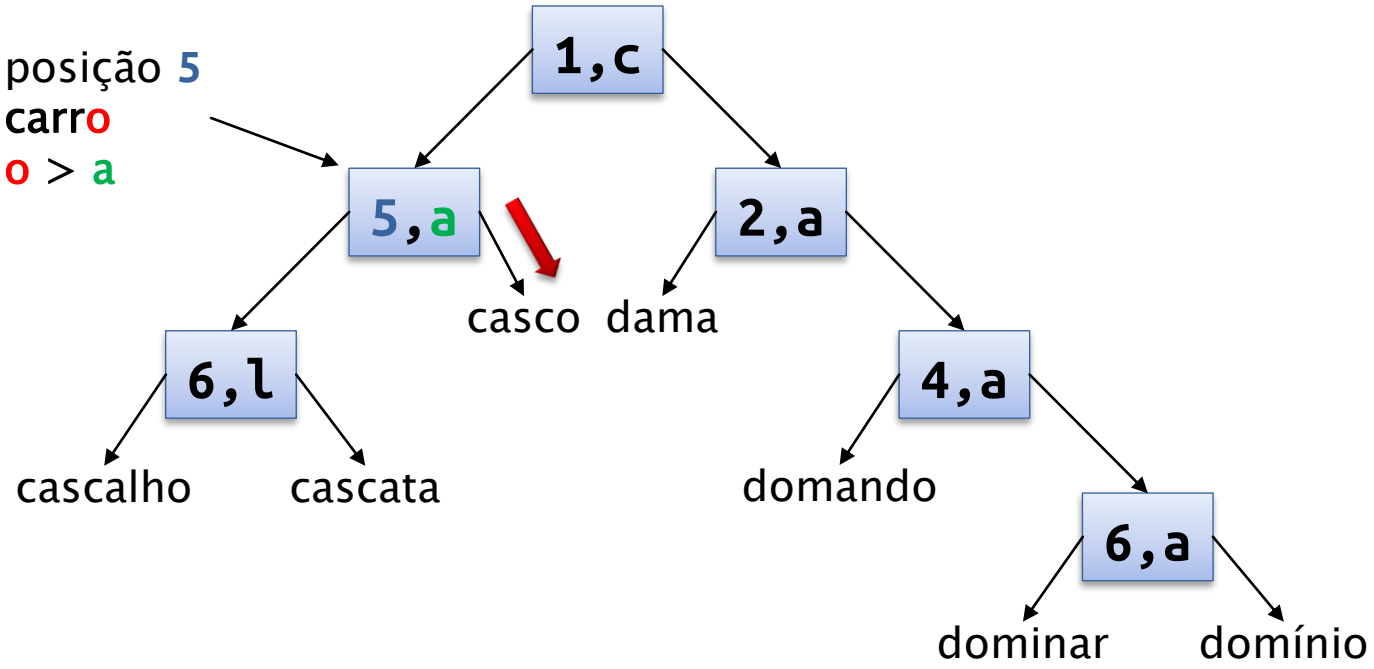
4.1. Busca

- Exemplo 2: buscar por **carro**



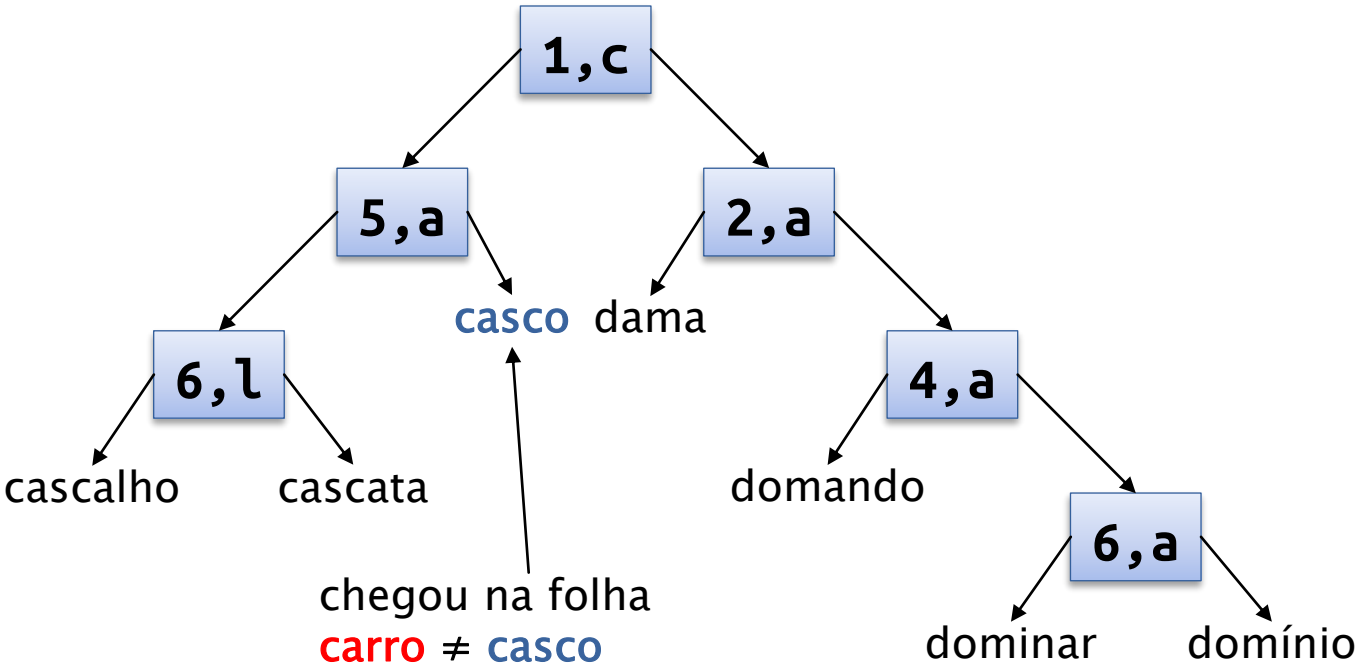
4.1. Busca

- Exemplo 2: buscar por **carro**



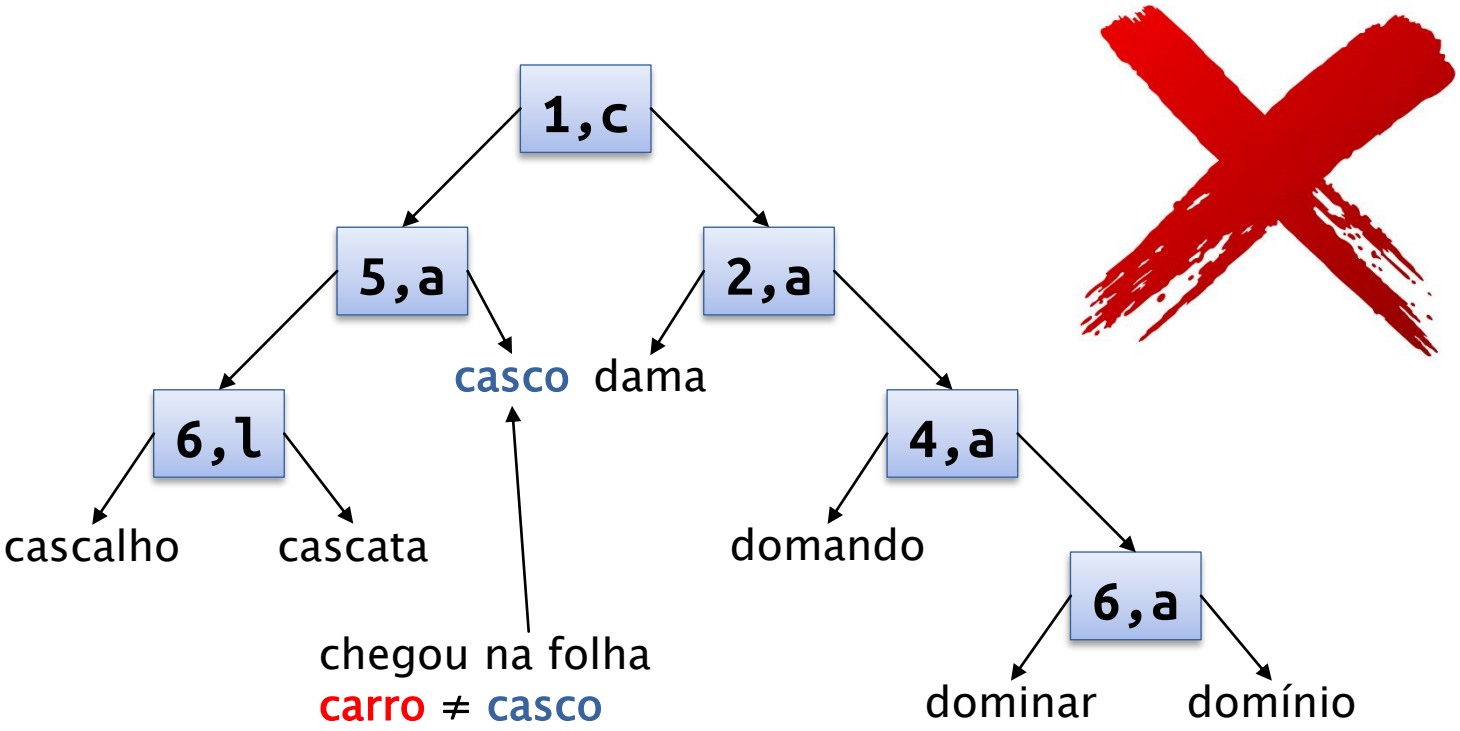
4.1. Busca

- Exemplo 2: buscar por **carro**



4.1. Busca

- Exemplo 2: buscar por **carro**



4.2. Inserção

- **Sequencia de passos:**

1. Se a sub-árvore atual for vazia, é criado um nó de informação com a chave **X** (isto ocorre somente na inserção da primeira chave) e o algoritmo termina.

2. Se a sub-árvore atual for simplesmente um nó de informação, os valores da chave **X** são comparados, a partir da posição de índice imediatamente após o último índice da sequência de índices consecutivos do caminho de pesquisa, com os índices correspondentes da chave **X** deste nó de informação, até encontrar um índice **i** cujos valores sejam diferentes.

   - A comparação dos valores a partir do último índice consecutivo melhora o desempenho do algoritmo: se todos forem iguais, a chave já se encontra na árvore e o algoritmo termina; senão, vai para o passo 4.

4.2. Inserção

- **Sequencia de passos:**

3. Se a raiz da sub-árvore atual for um nó de desvio, deve-se prosseguir para a sub-árvore indicada pelo valor da chave **X** de índice dado pelo nó atual, de forma recursiva.

4. Criar um nó de desvio e um nó de informação: o primeiro contendo o índice **i** e o valor da posição onde existe a diferença e o segundo a chave **X**. A seguir, o nó de desvio é ligado ao de informação pelo ponteiro de sub-árvore esquerda ou direita, dependendo se o valor de índice i da chave X seja maior ou menor.

5. O caminho de inserção é percorrido novamente de baixo para cima, subindo com o par de nós criados no passo 4 até chegar a um nó de desvio cujo índice seja menor que o índice **i** determinado no passo 2: este é o ponto de inserção e o par de nós é inserido.

4.2. Inserção

- Exemplo: inserir **consultorio**

4.2. Inserção

- Exemplo: inserir **consultorio**

consultorio

4.2. Inserção

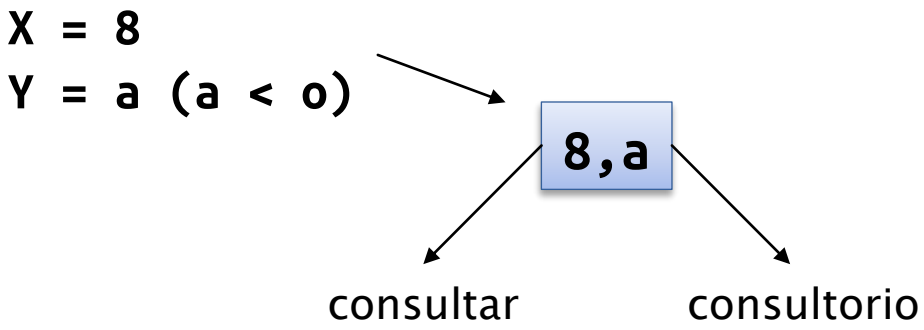
- Exemplo: inserir **consultar**

consultorio

consultorio, consultar
Diferença no 8º caractere

4.2. Inserção

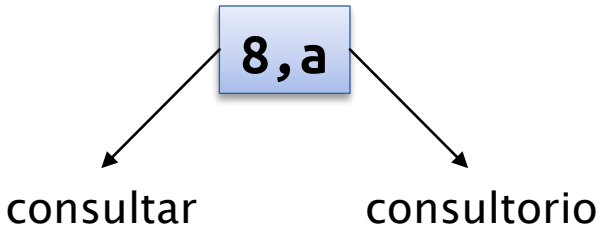
- Exemplo: inserir **consultar**



consultorio, consultar
Diferença no 8º caractere

4.2. Inserção

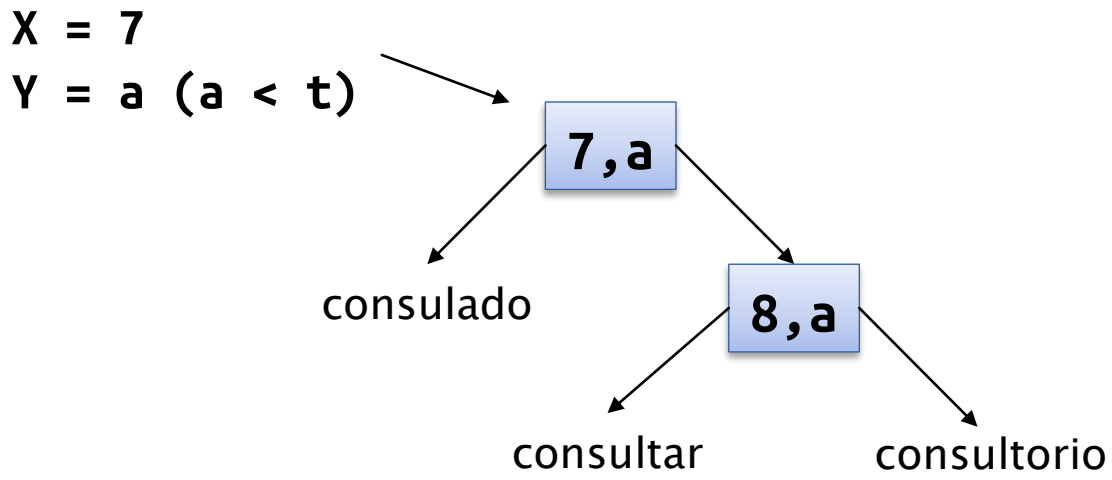
- Exemplo: inserir **consulado**



consultar, consulado
Diferença no 7º caractere

4.2. Inserção

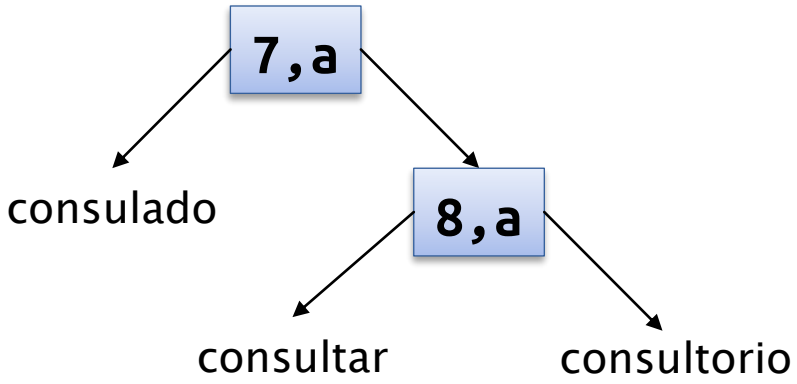
- Exemplo: inserir **consulado**



consultar, consulado
Diferença no 7º caractere

4.2. Inserção

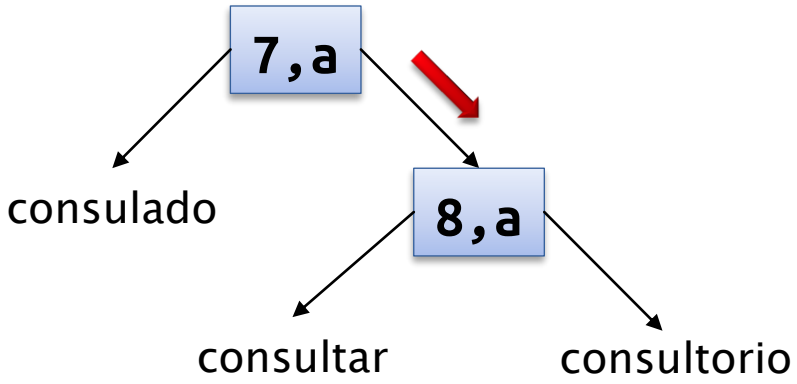
- Exemplo: inserir **consultado**



consulado, consultado
t > **a** no 7º caractere

4.2. Inserção

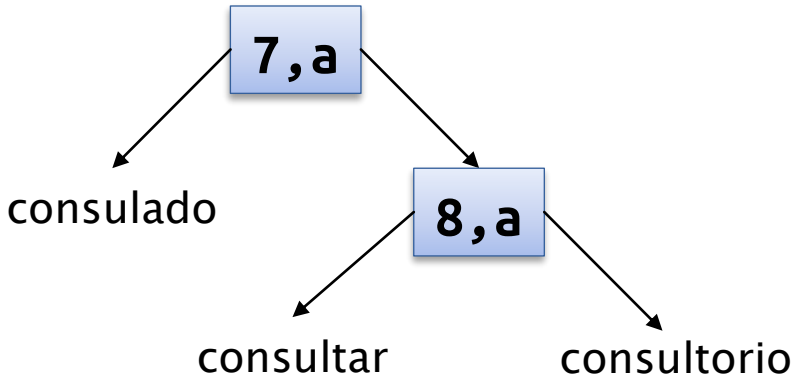
- Exemplo: inserir **consultado**



consulado, consultado
t > **a** no 7º caractere

4.2. Inserção

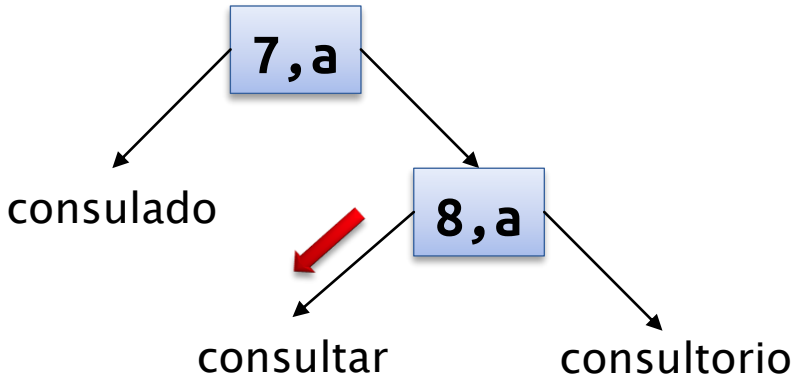
- Exemplo: inserir **consultado**



consultar, consultado
a = a no 8º caractere

4.2. Inserção

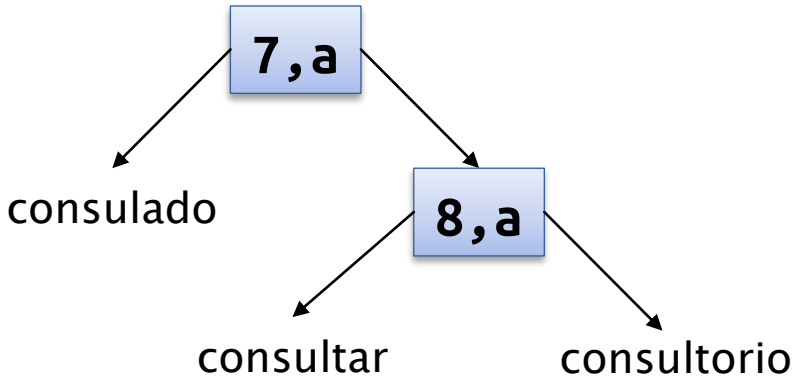
- Exemplo: inserir **consultado**



consultar, consultado
a = **a** no 8º caractere

4.2. Inserção

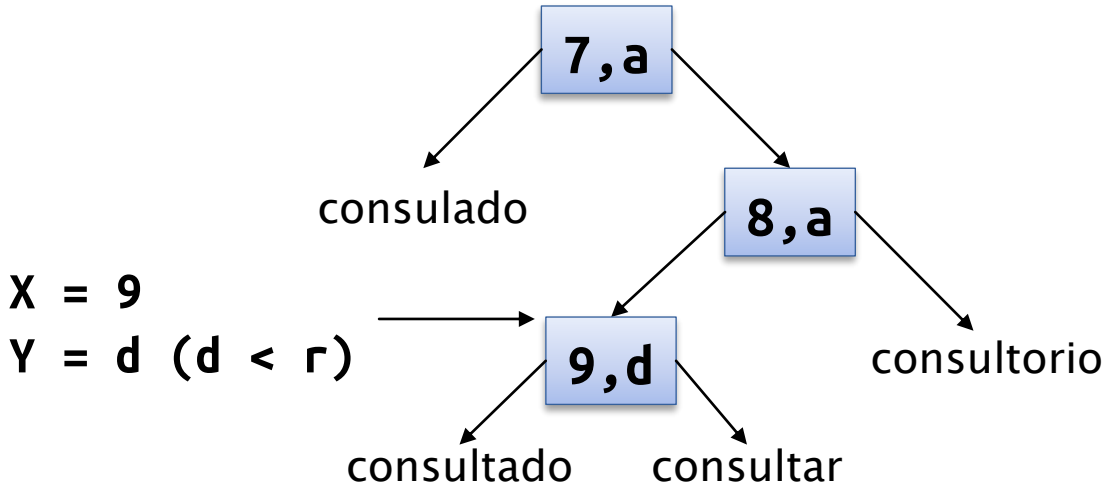
- Exemplo: inserir **consultado**



consultar, consultado
Diferença no 9º caractere

4.2. Inserção

- Exemplo: inserir **consultado**



consultar, consultado
Diferença no 9º caractere

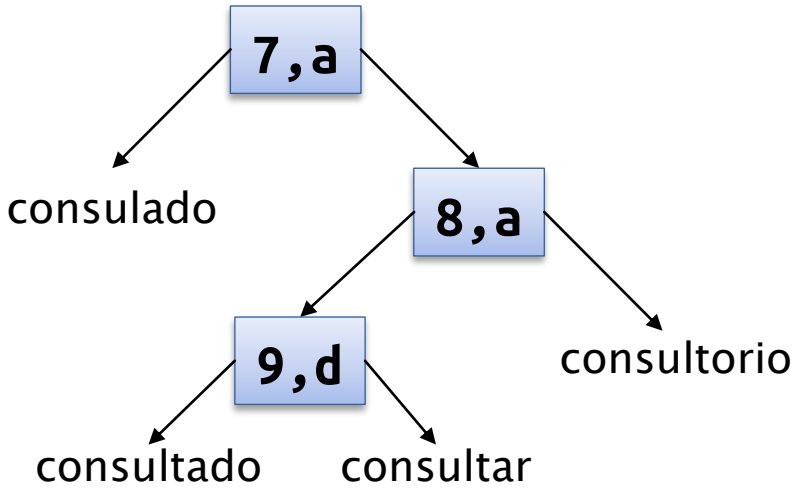
4.3. Remoção

- **Sequencia de Passos:**

1. Buscar e apagar a chave da árvore.
2. O pai da chave deve ser apagado.

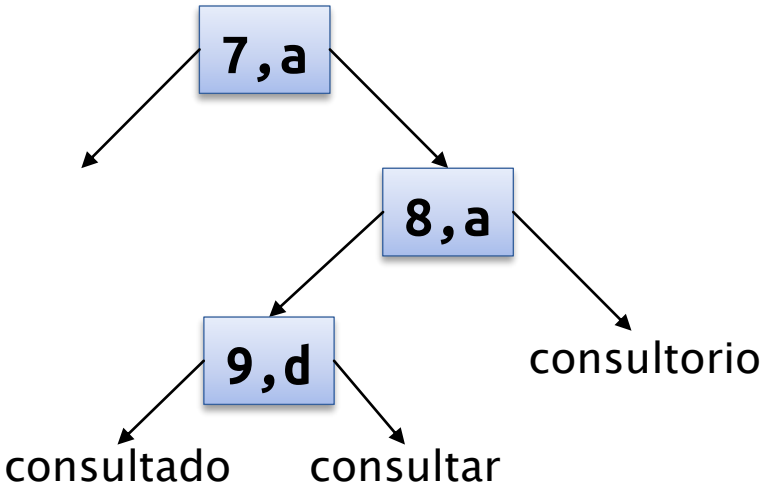
4.3. Remoção

- Exemplo: remover **consulado**



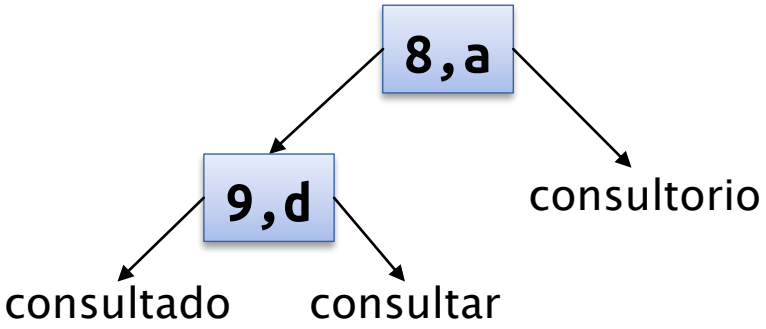
4.3. Remoção

- Exemplo: remover **consulado**



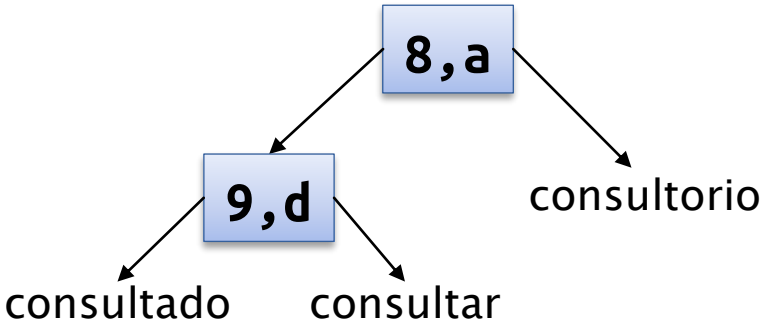
4.3. Remoção

- Exemplo: remover **consulado**



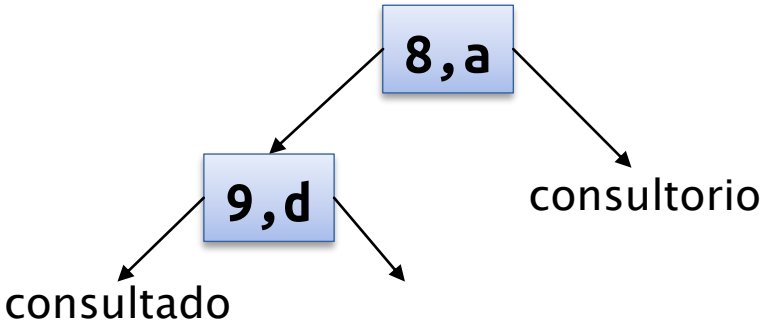
4.3. Remoção

- Exemplo: remover **consultar**



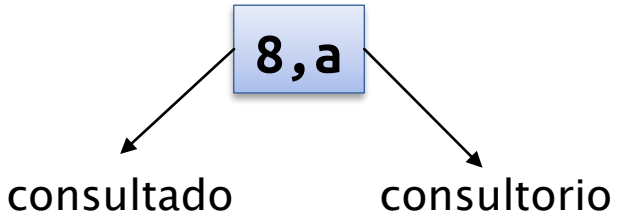
4.3. Remoção

- Exemplo: remover **consultar**



4.3. Remoção

- Exemplo: remover **consultar**



4.4. Resumo

- A pesquisa digital está baseada na representação das chaves como uma sequência de caracteres ou de dígitos.

- Os métodos de pesquisa digital são particularmente vantajosos quando as chaves são grandes e de tamanho variável.

5. Referências

- Material de aula dos Profs. Luiz Chaimowicz e Raquel O. Prates, da UFMG: https://homepages.dcc.ufmg.br/~glpappa/aeds2/AEDS2.1%20Conceitos%20Basicos%20TAD.pdf
- Horowitz, E. & Sahni, S.; Fundamentos de Estruturas de Dados, Editora Campus, 1984.
- Wirth, N.; Algoritmos e Estruturas de Dados, Prentice/Hall do Brasil, 1989.
- Material de aula do Prof. José Augusto Baranauskas, da USP: https://dcm.ffclrp.usp.br/~augusto/teaching.htm
- Material de aula do Prof. Rafael C. S. Schouery, da Unicamp: https://www.ic.unicamp.br/~rafael/cursos/2s2019/mc202/index.html