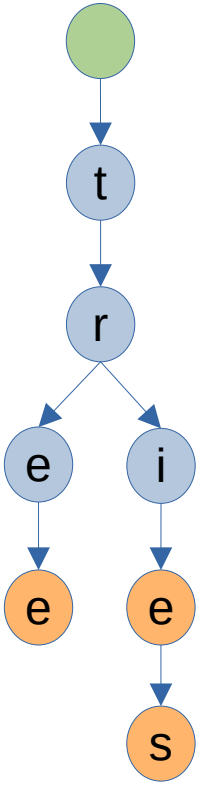
Pesquisa e Ordenação de Dados

Unidade 5.4:

Tries



Busca

- Seja uma busca de uma chave **x** em um conjunto de chaves **S={s_1,...,s_n}**.
  - Os métodos de busca já vistos organizam **S** de modo a possibilitar encontrar **x** através de comparações de igualdade entre cada **s_i** e **x**.
  - As chaves **s_i** e **x** são tratadas como indivisíveis.
- Estes são métodos ideais para consultas por equivalência, mas inviáveis para outros tipos de consultas (como consultas por similaridade ou por substrings).
  - **Busca digital**: métodos que comparam cada bit (ou dígito) de **s_i** e **x.**

Busca Digital

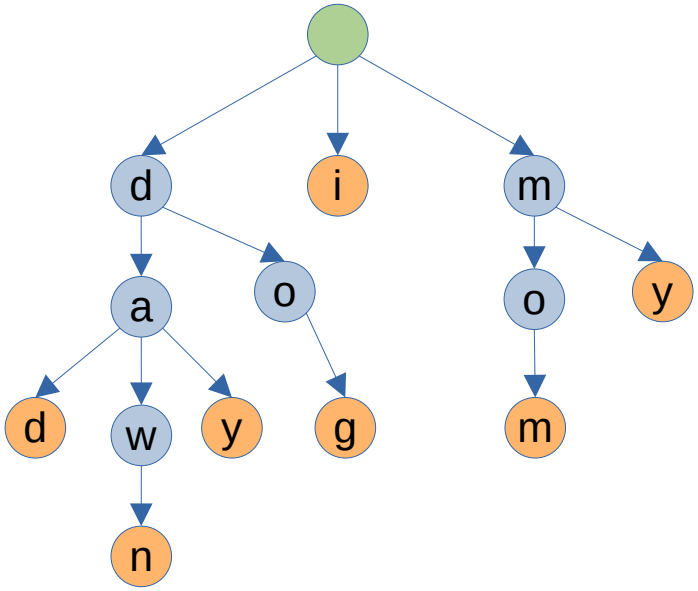
- Na busca digital, a chave é tratada como um elemento **divisível**.
  - Cada chave é formada por um conjunto de caracteres pertencentes a um alfabeto de símbolos
  - Exemplos de alfabetos:
    - {0,1}, {A,B,C,...Z, a, b, c,..., z}, {0,1,2,3,4,5,...,9}
  - Exemplos de chaves:
    - 01010101000000000010100000001010
    - ABABBBABABA, Maria
    - 19034717
  - Na busca, a comparação é efetuada entre os dígitos que compõem as chaves (dígito a dígito, caractere a caractere).

Trie

- trie = retrieval (recuperação)
- Tipo de **árvore** de busca utilizada para armazenar e recuperar de forma eficiente um **conjunto** de chaves (normalmente **strings**)
- Ao invés de armazenar uma chave completa (como numa BST), cada nó de uma Trie (exceto a raiz) armazenará um único caractere
  - a chave completa será encontrada através do percurso a partir da raiz
  - utiliza parte da chave como caminho de busca
- Também chamada de **árvore de prefixos**, já que os descendentes de um nó possuem um prefixo comum

Trie

- Cada nodo representa apenas um caractere e pode conter múltiplos ramos, cada qual indicando um possível próximo caractere da chave.
- Para acessar uma chave, realizamos uma busca em profundidade a partir da raiz, seguindo os links entre os nodos
- Cada nível percorrido corresponde a avançar um dígito na chave
  - Nodos marcados como terminais indicam que aquele percurso forma uma chave completa.
  - Nodos não terminais indicam um prefixo de uma ou mais chaves.



Chaves indexadas:
dad
dawn
day
dog
I
mom
my

Trie - Características

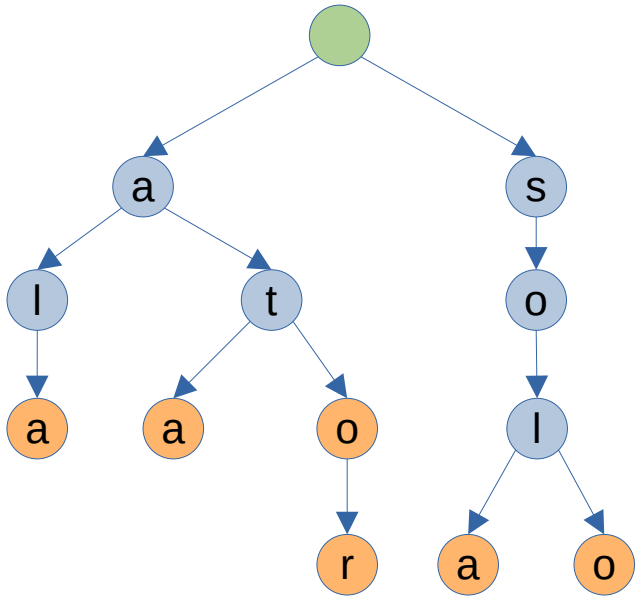
- Árvore n-ária: o grau da árvore corresponde ao tamanho do alfabeto;
- A altura da árvore é igual ao comprimento da chave mais longa;
- O nó raiz representa uma chave vazia;
- O caminho da raiz para qualquer nó é um prefixo de uma chave indexada; ou seja, os descendentes do mesmo nó têm o mesmo prefixo;
- Os nós internos formam os prefixos das chaves de busca indexadas, além de indexar outros nós que aumentam o prefixo;
- Os nós folha e os nós internos sinalizados formam uma chave completa. Esta sinalização pode ser um flag ou algum valor associado à chave;
- O formato das tries não depende da ordem em que as chaves são inseridas, mas apenas dos valores das chaves.

Operações sobre Tries

- **Busca**: inicia pela raiz e pelo primeiro caractere da chave procurada
  1) Se o caractere não pertence à árvore, então a chave não pertence à trie;
  2) Se o caractere pertence à árvore e o nó atual não é terminal:
     - Se existe um filho que corresponda ao próximo caractere da chave, avance um caractere e visite o respectivo filho (volte para o passo 2);
     - Se não existe esse filho, então a chave não está indexada.
  3) Se o caractere pertence a árvore e o nó atual é terminal: então esse nó representa a resposta.

Operações sobre Tries

- Considere a seguinte trie:



raiz

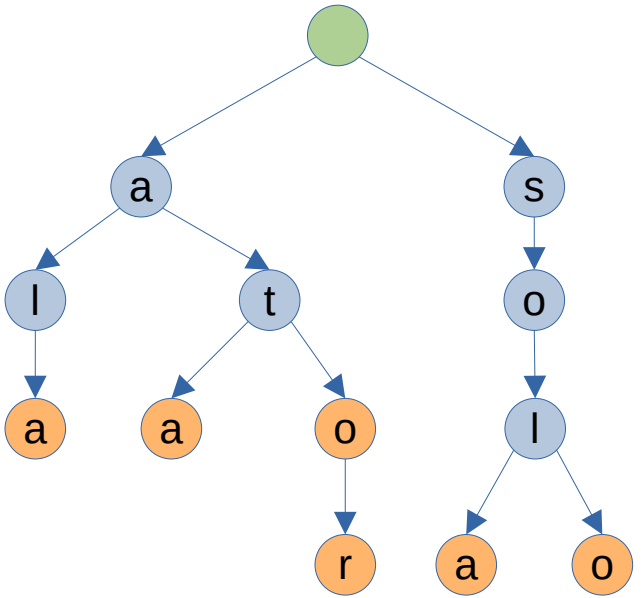
nó intermediário

nó terminal

Chaves:
ala
ata
ato
ator
sola
solo

Operações sobre Tries

- Busca: galo

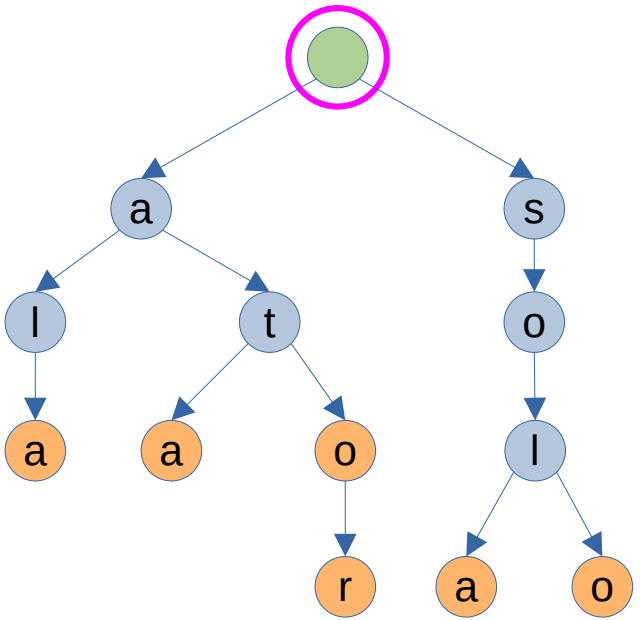


a raiz não possui filho com o caractere g.

Chave não encontrada!

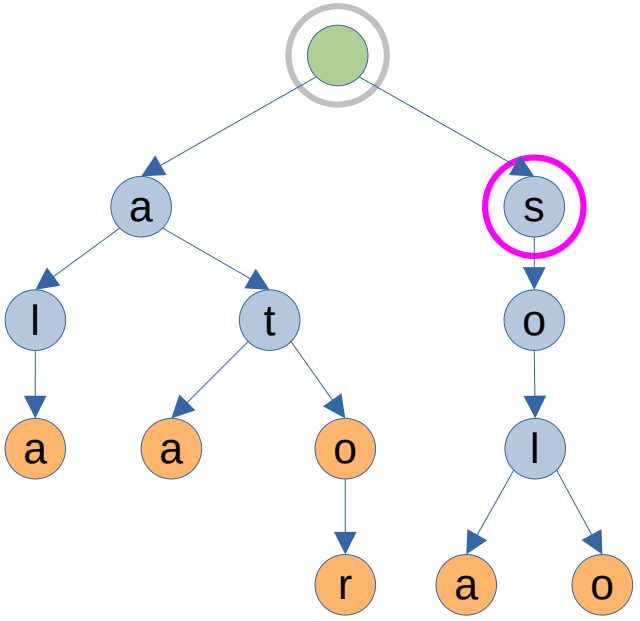
Operações sobre Tries

- Busca: sol



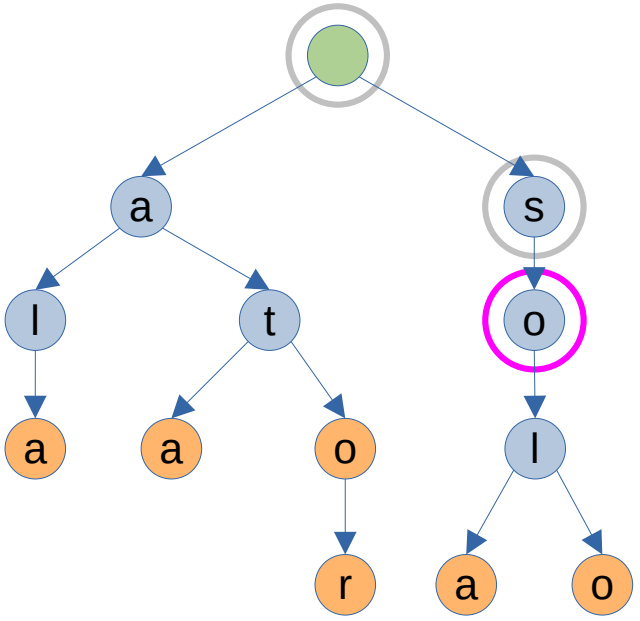
Operações sobre Tries

- Busca: sol



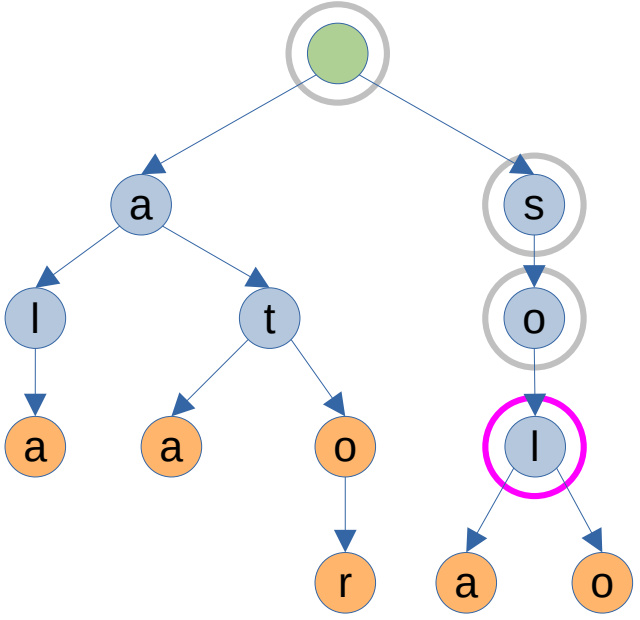
Operações sobre Tries

- Busca: sol



Operações sobre Tries

- Busca: sol

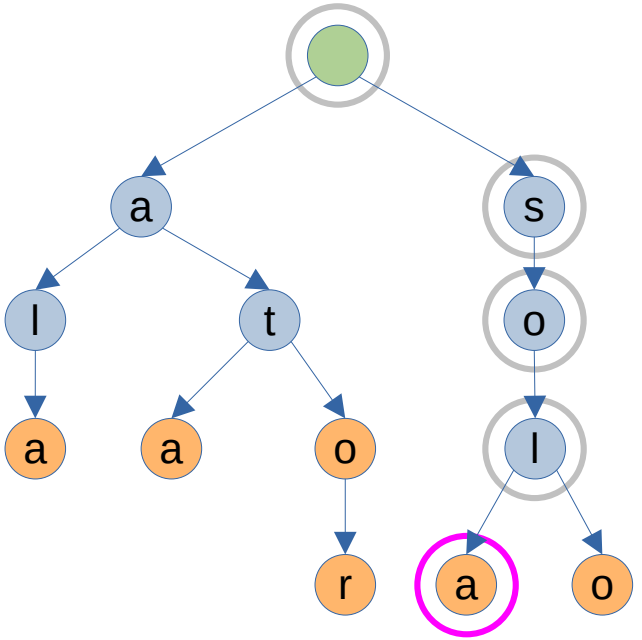


Nó não é terminal

Chave não encontrada!

Operações sobre Tries

- Busca: sola



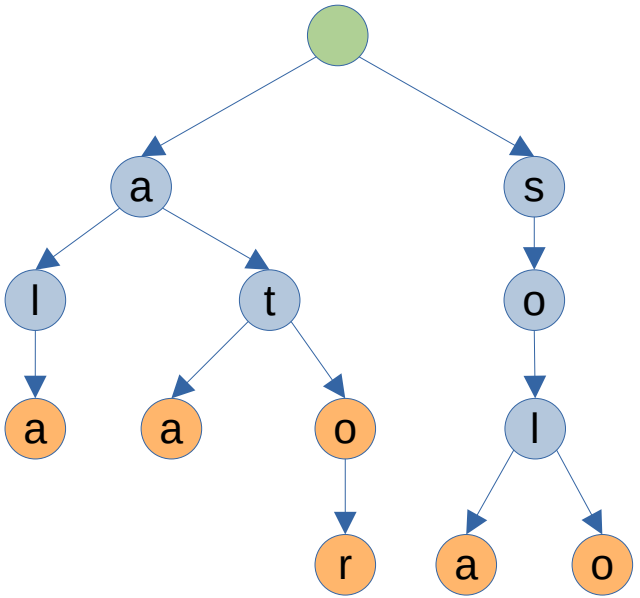
Chave encontrada!

Operações sobre Tries

- **Inserção**:
  1) Faz-se uma busca pela palavra a ser inserida;
  2) Se ela já existir na trie nada é feito (ou quando houver um valor associado, este pode ser sobrescrito, dependendo da implementação);
  3) Caso contrário, é recuperado o nó **n** até onde acontece a maior substring da palavra a ser inserida (último nó visitado);
  4) O restante dos caracteres (quando existentes) da nova chave são adicionados, a partir do nó **n** (um caractere por nó);
  5) O nó correspondente ao último caractere é marcado como terminal.

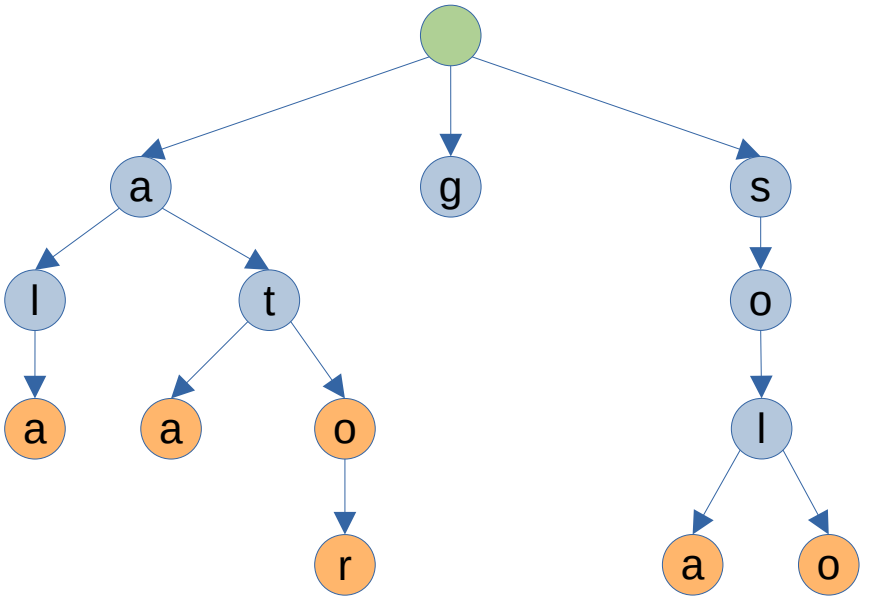
Operações sobre Tries

- Inserir: galo



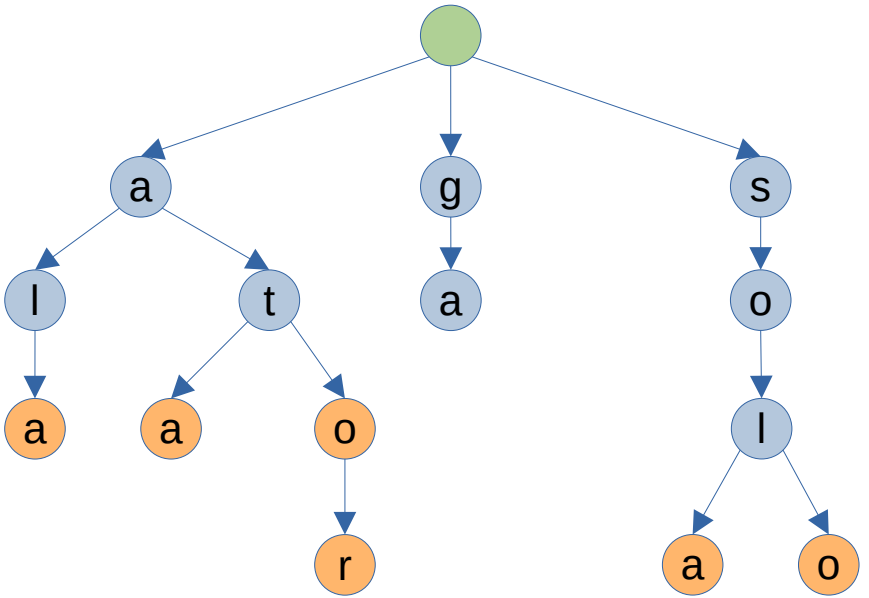
Operações sobre Tries

- Inserir: galo



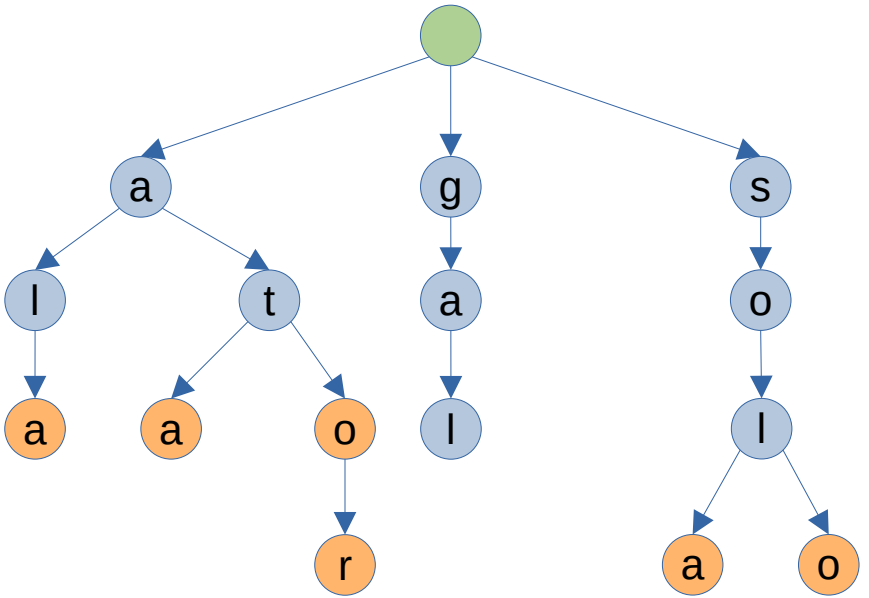
Operações sobre Tries

- Inserir: galo



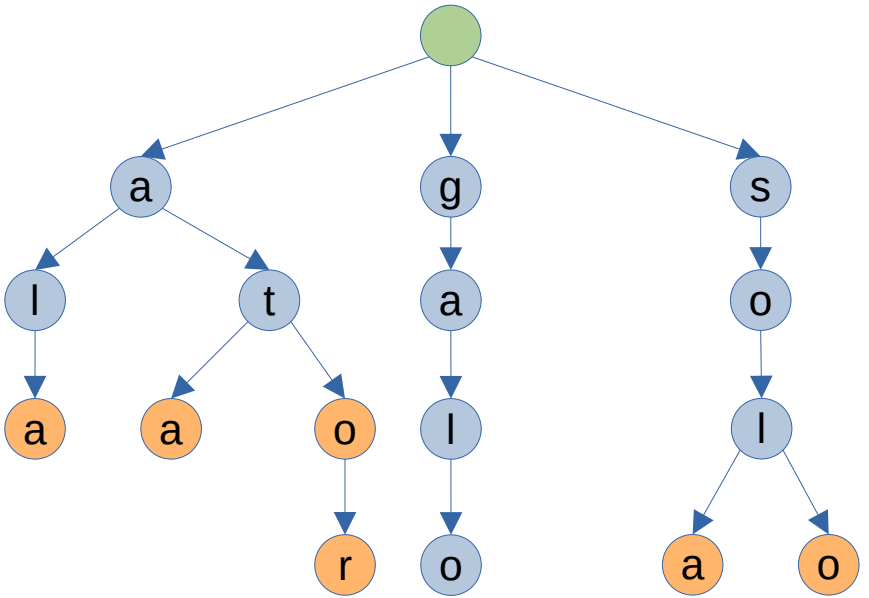
Operações sobre Tries

- Inserir: galo



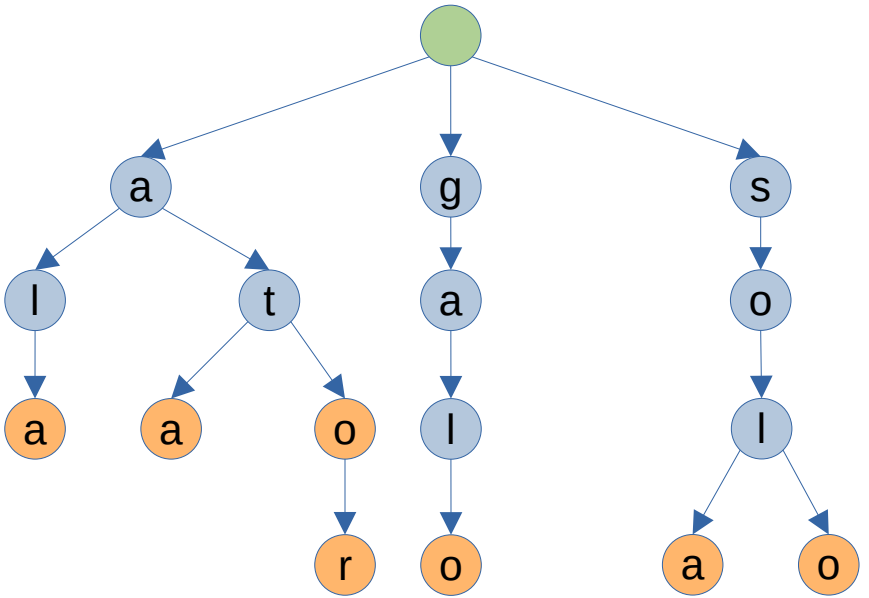
Operações sobre Tries

- Inserir: galo



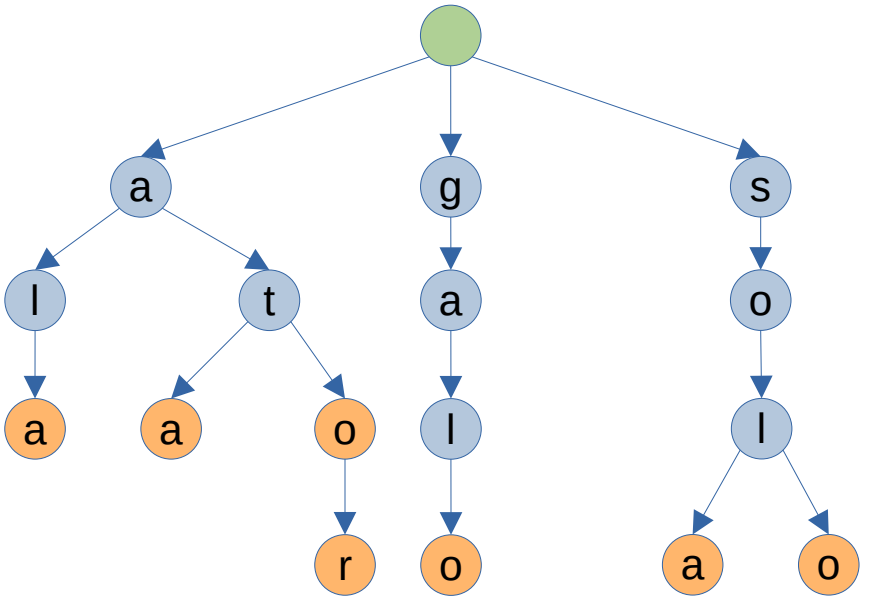
Operações sobre Tries

- Inserir: galo



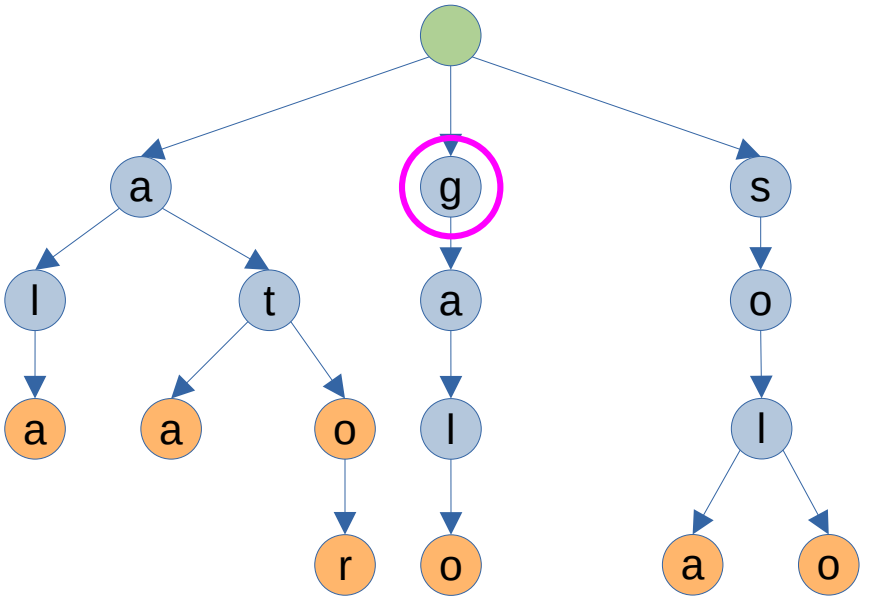
Operações sobre Tries

- Inserir: gel



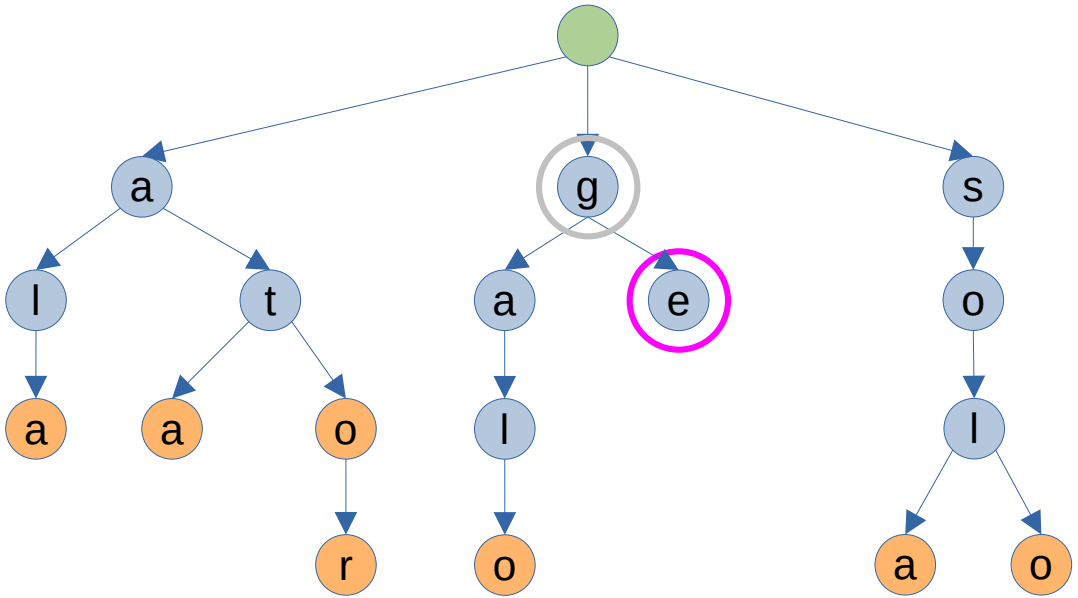
Operações sobre Tries

- Inserir: gel



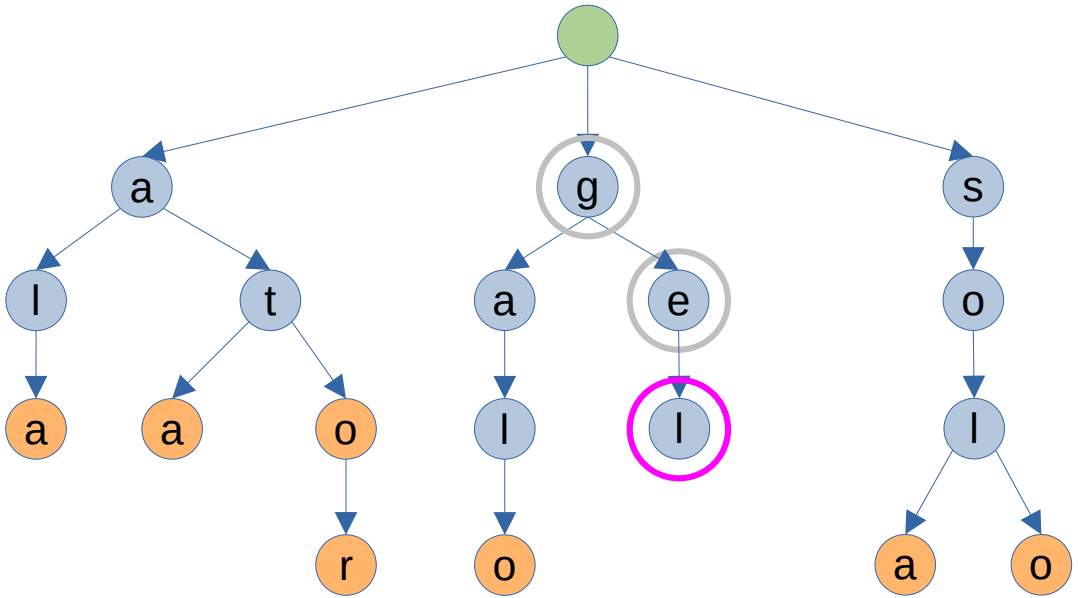
Operações sobre Tries

- Inserir: gel



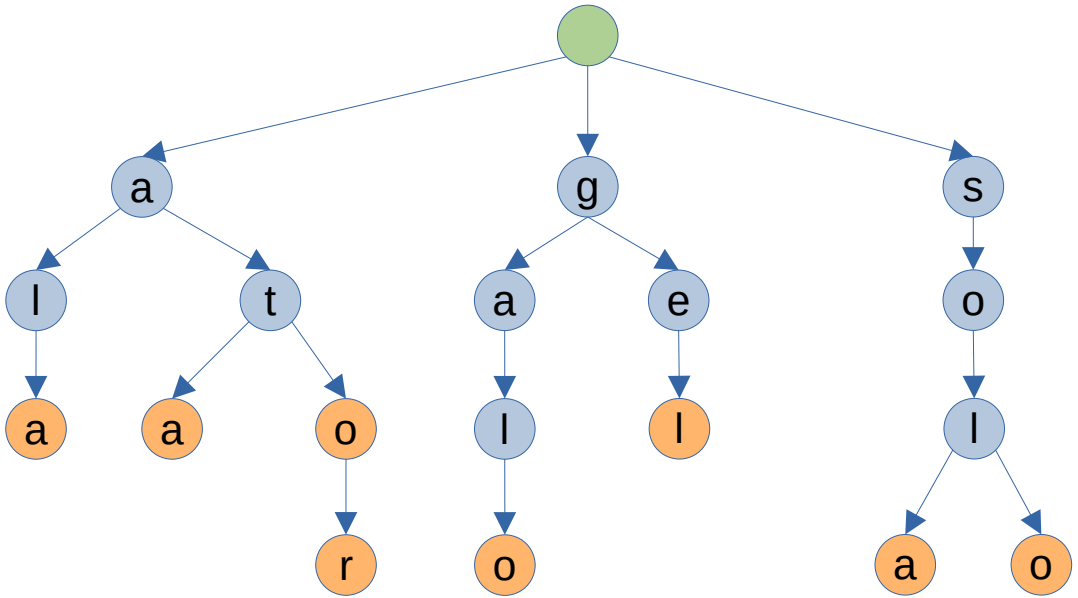
Operações sobre Tries

- Inserir: gel



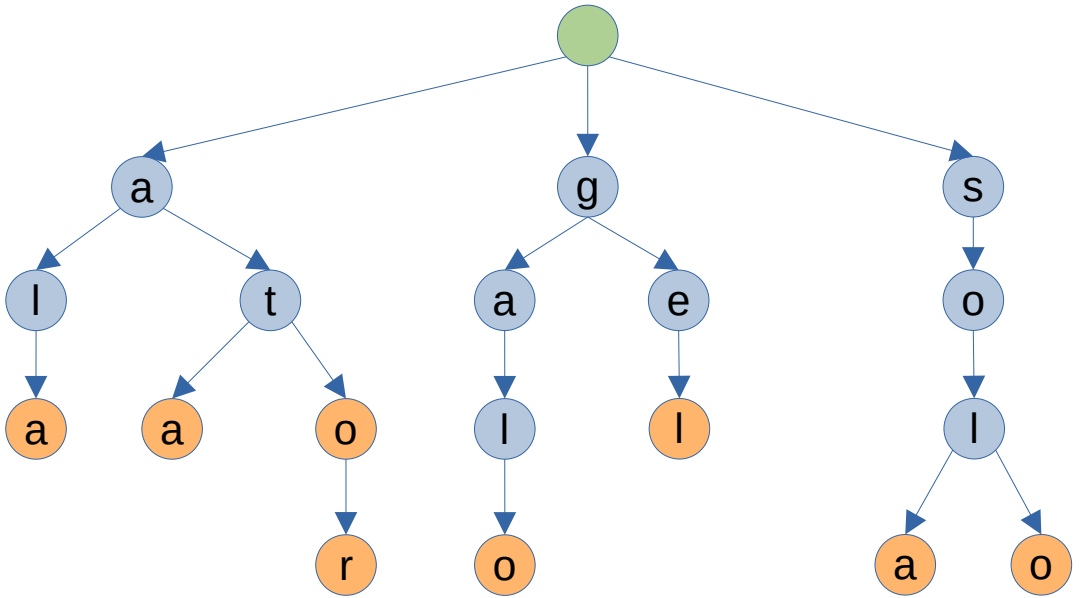
Operações sobre Tries

- Inserir: gel



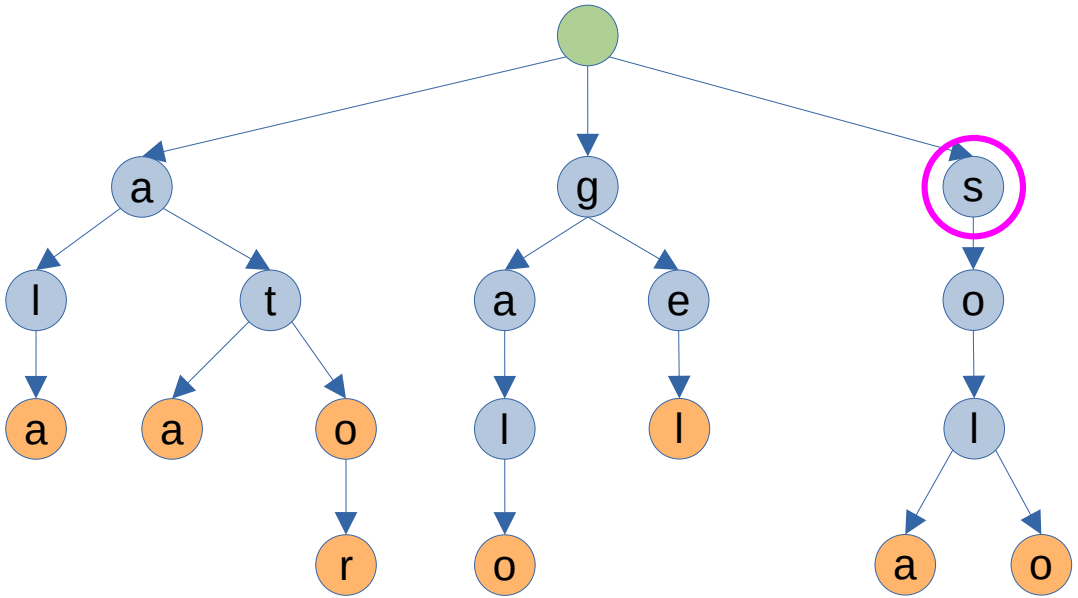
Operações sobre Tries

- Inserir: sol



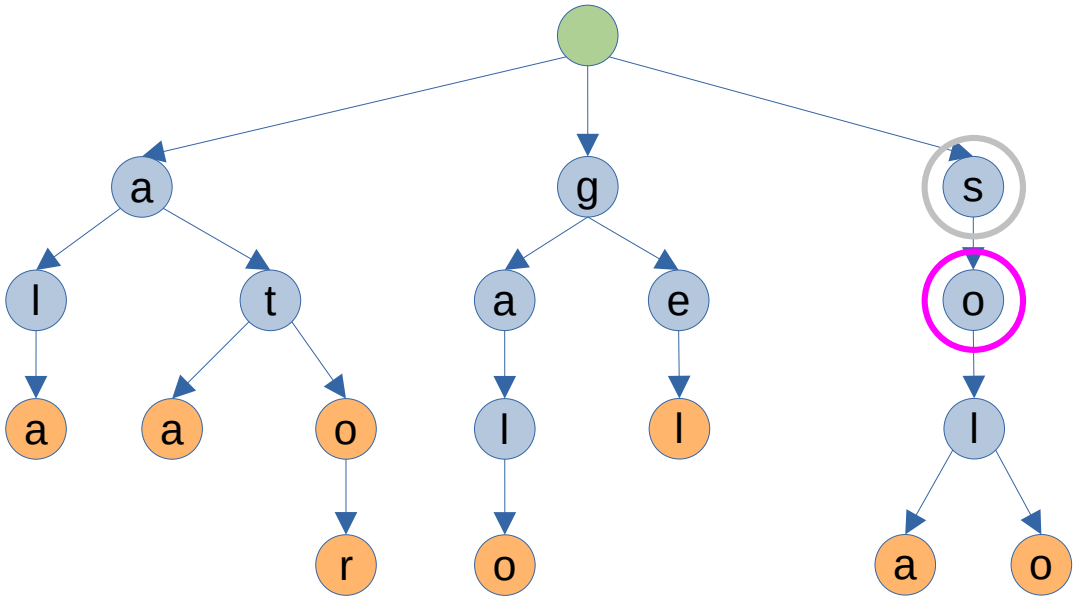
Operações sobre Tries

- Inserir: sol



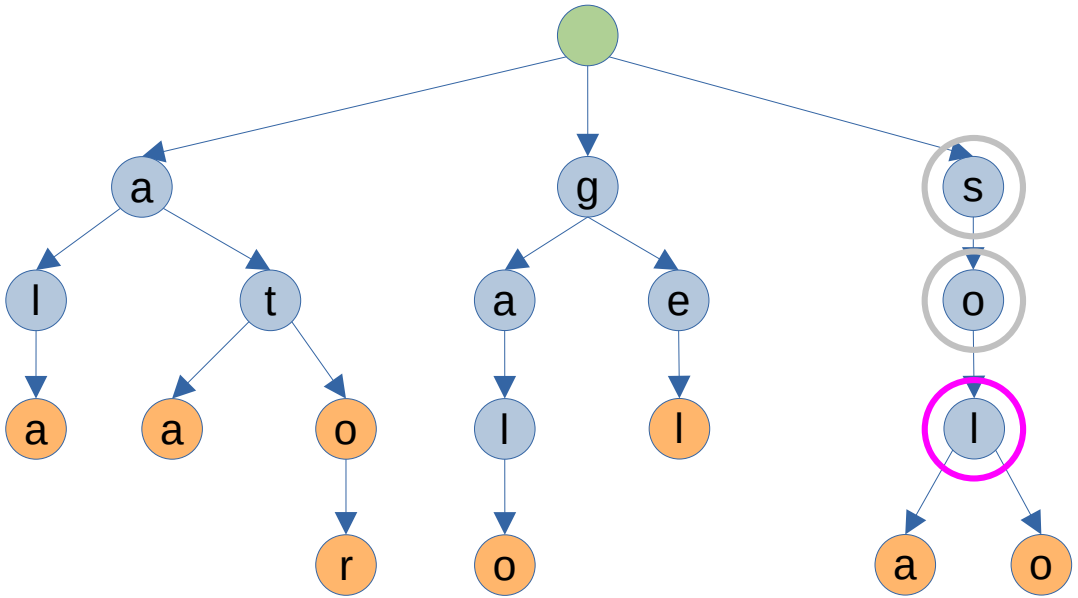
Operações sobre Tries

- Inserir: sol



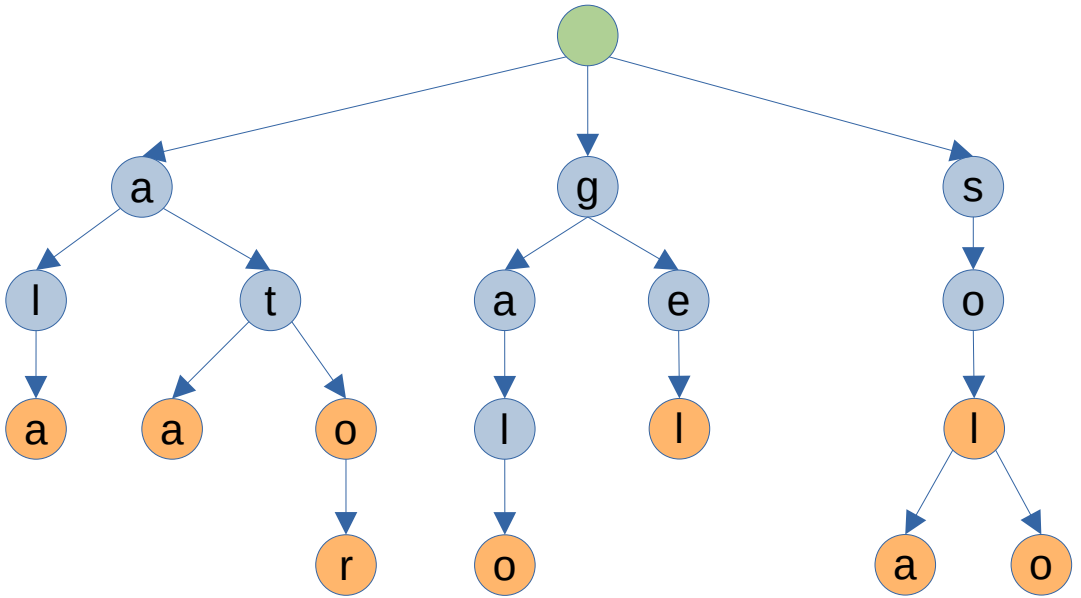
Operações sobre Tries

- Inserir: sol



Operações sobre Tries

- Inserir: sol

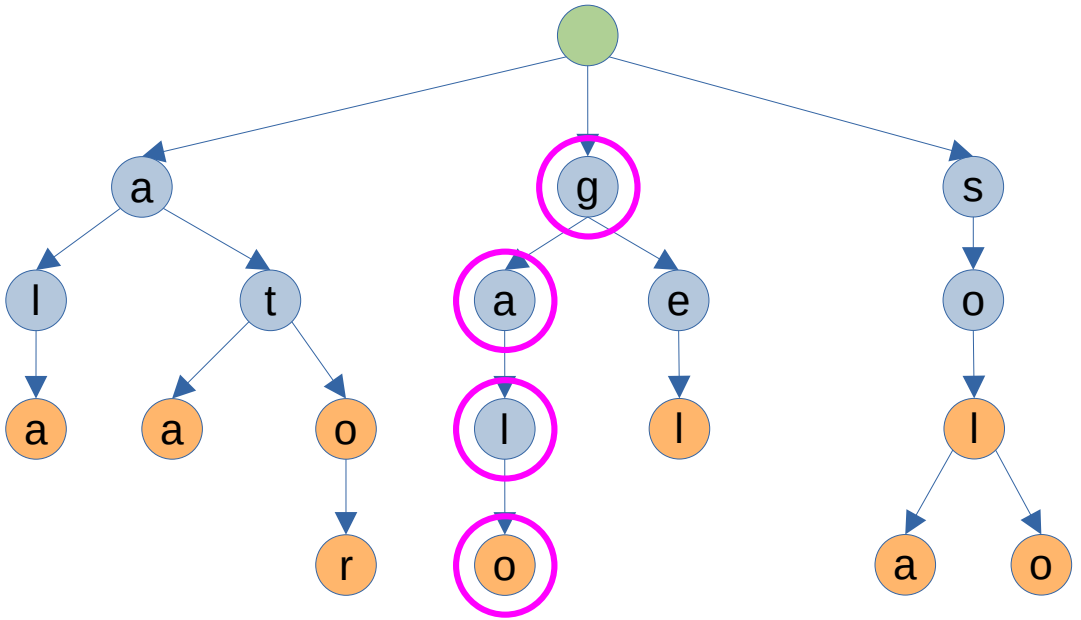


Operações sobre Tries

- Remoção:
  - Busca-se o nó que representa o final da chave a ser removida.
    - se este nó possui filho(s), torná-lo não terminal
    - se não possui filhos, a partir da folha e pelo caminho ascendente (*bottom-up*), são removidos todos os nós que têm apenas um filho e que não são terminais. A remoção é concluída quando se encontra um nó com mais de um filho ou um nó terminal.

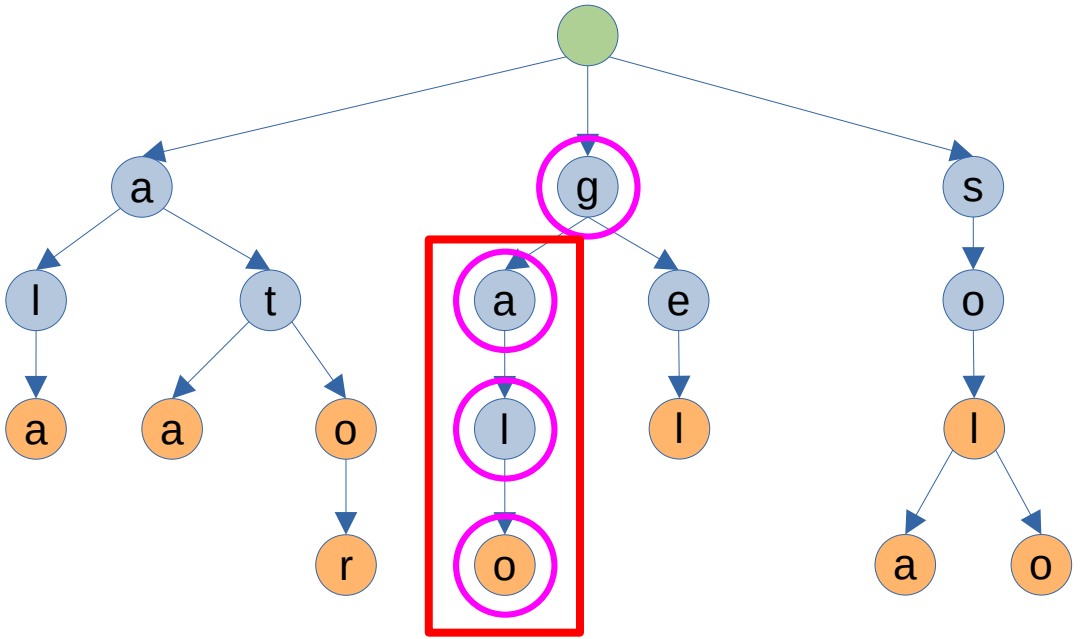
Operações sobre Tries

- Remover: galo



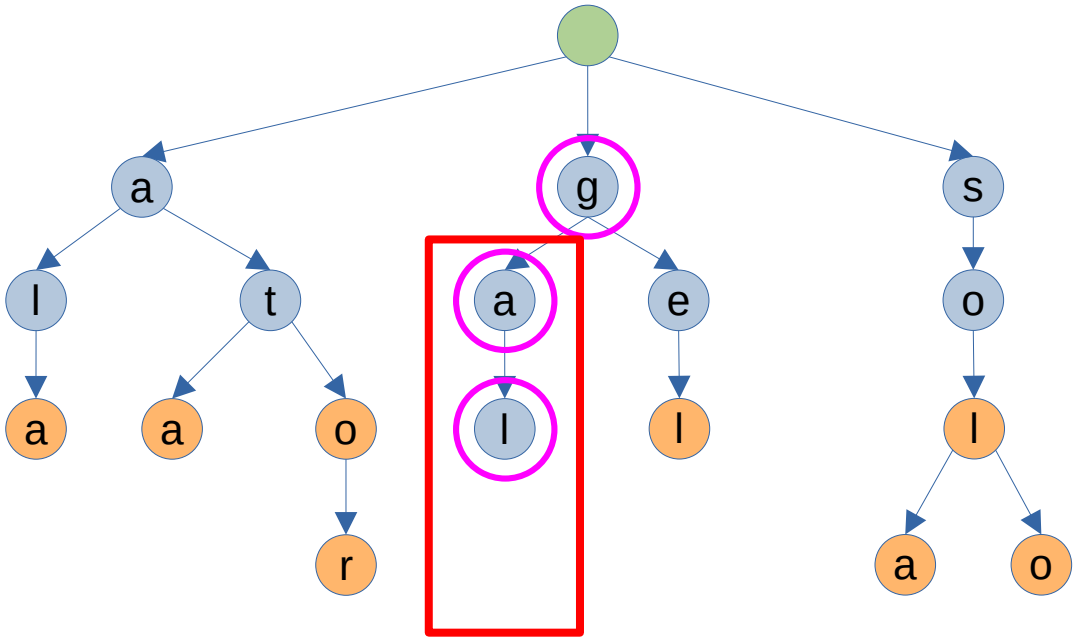
Operações sobre Tries

- Remover: galo



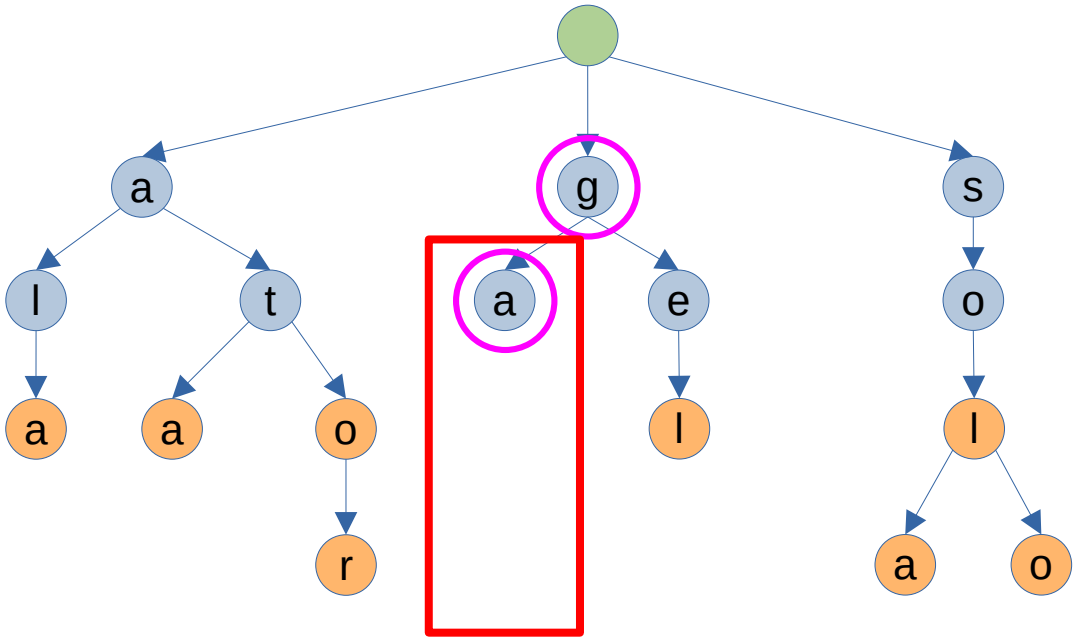
Operações sobre Tries

- Remover: galo



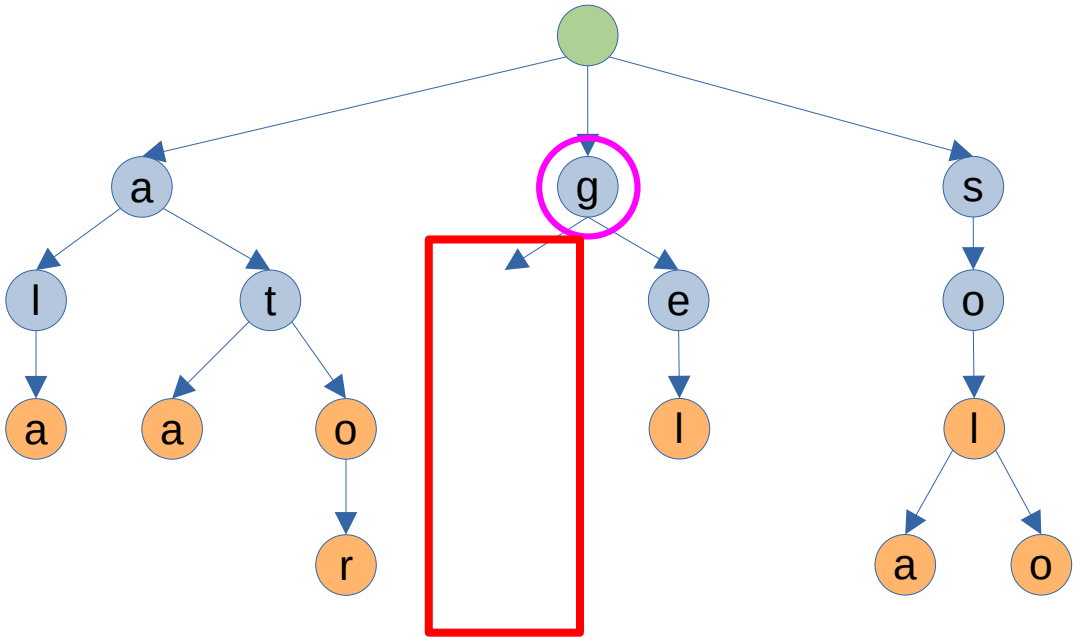
Operações sobre Tries

- Remover: galo



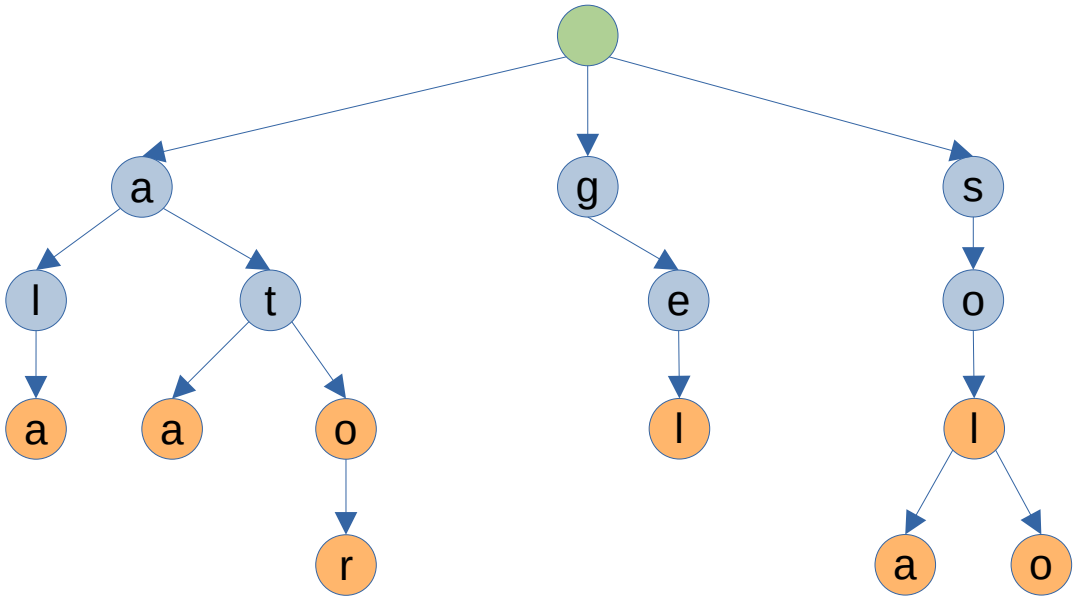
Operações sobre Tries

- Remover: galo



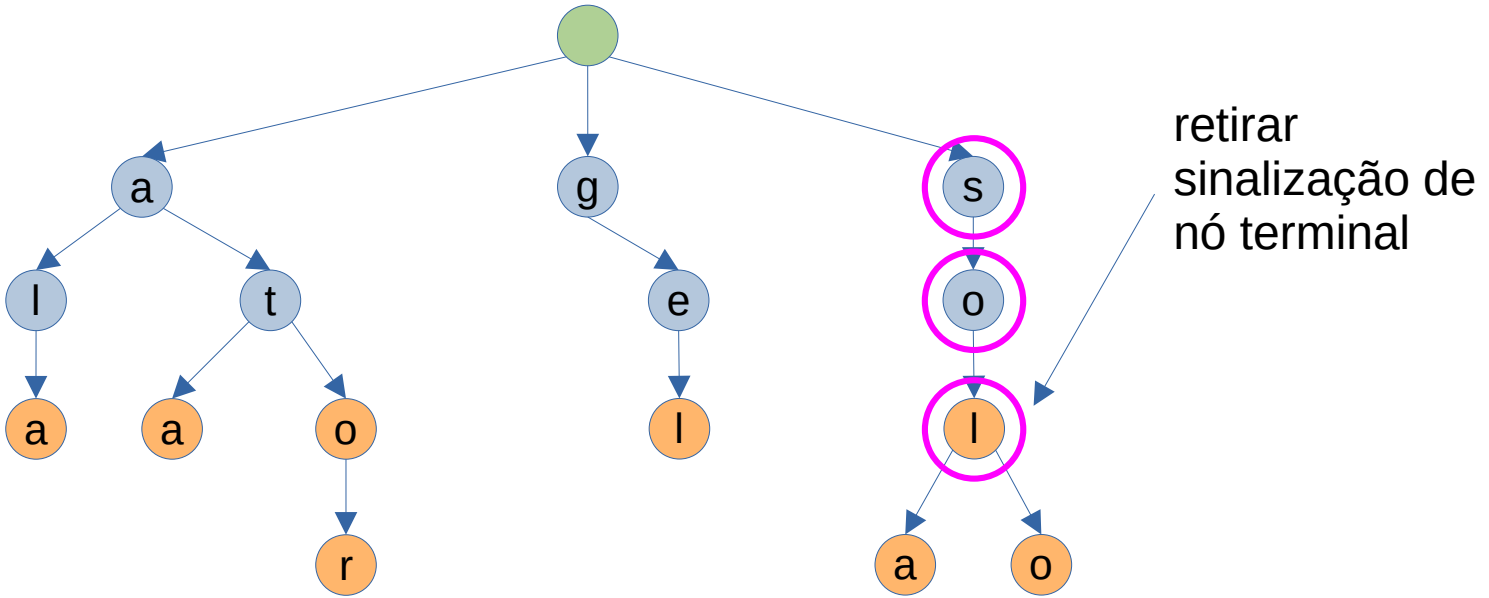
Operações sobre Tries

- Remover: galo



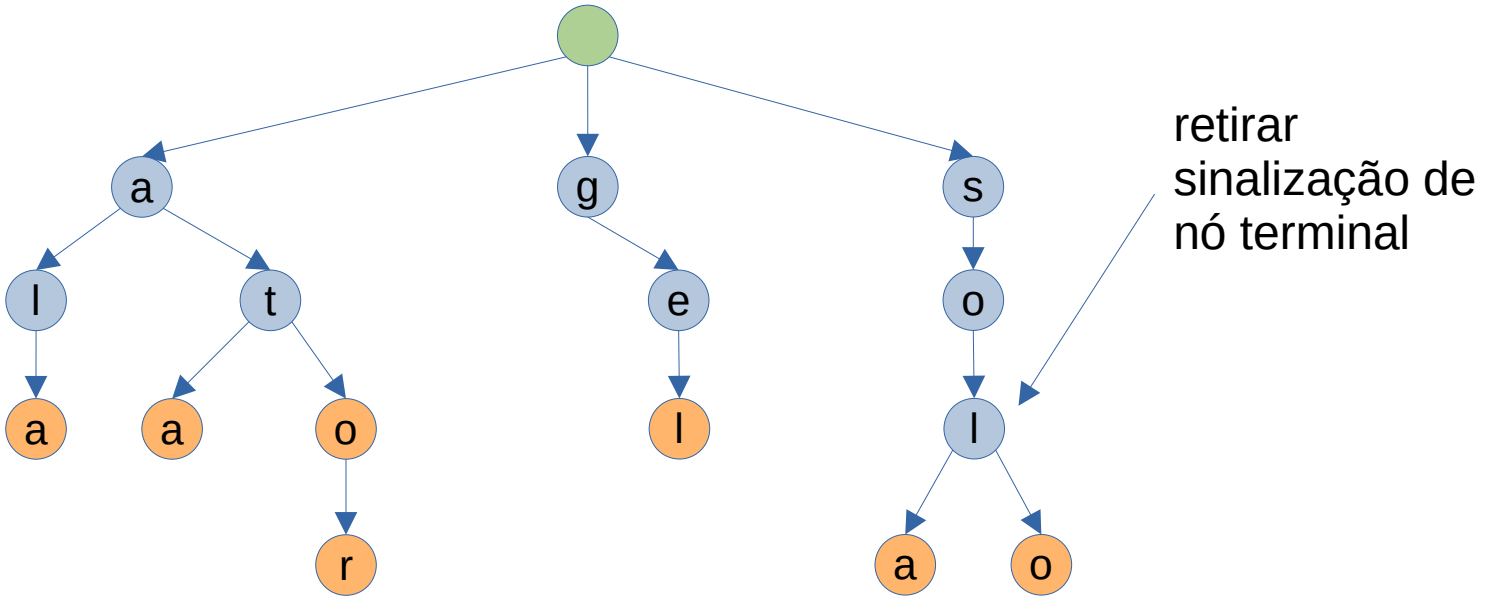
Operações sobre Tries

- Remover: sol



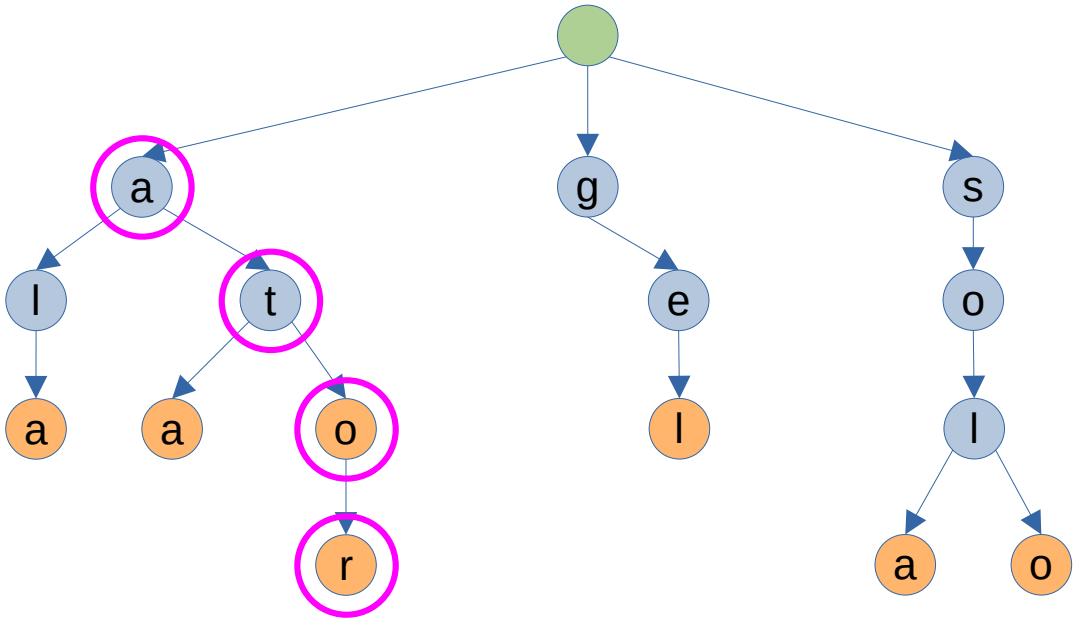
Operações sobre Tries

- Remover: so



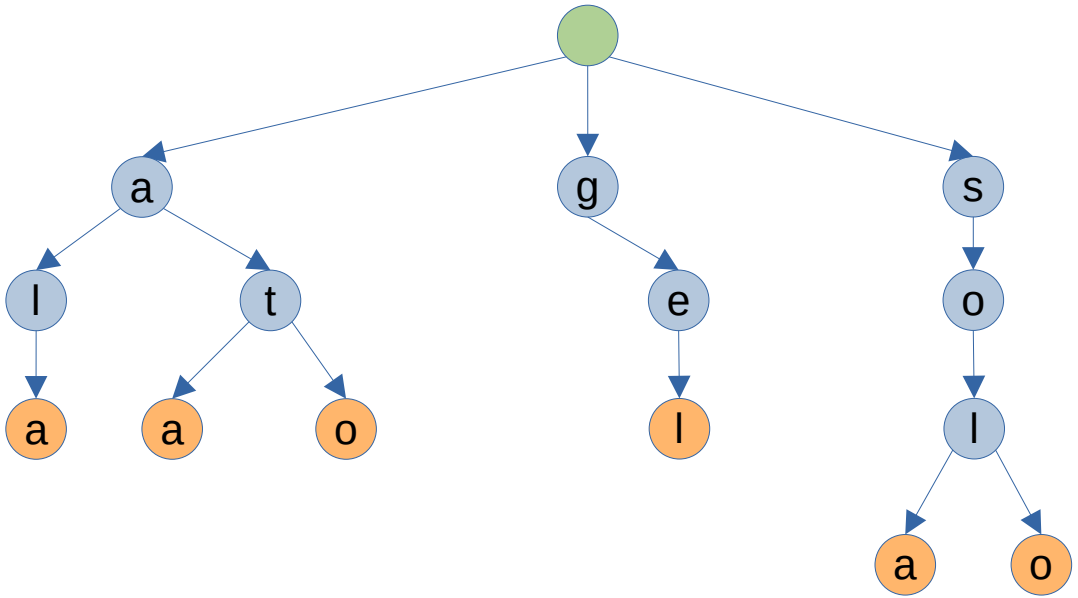
Operações sobre Tries

- Remover: ator



Operações sobre Tries

- Remover: ator



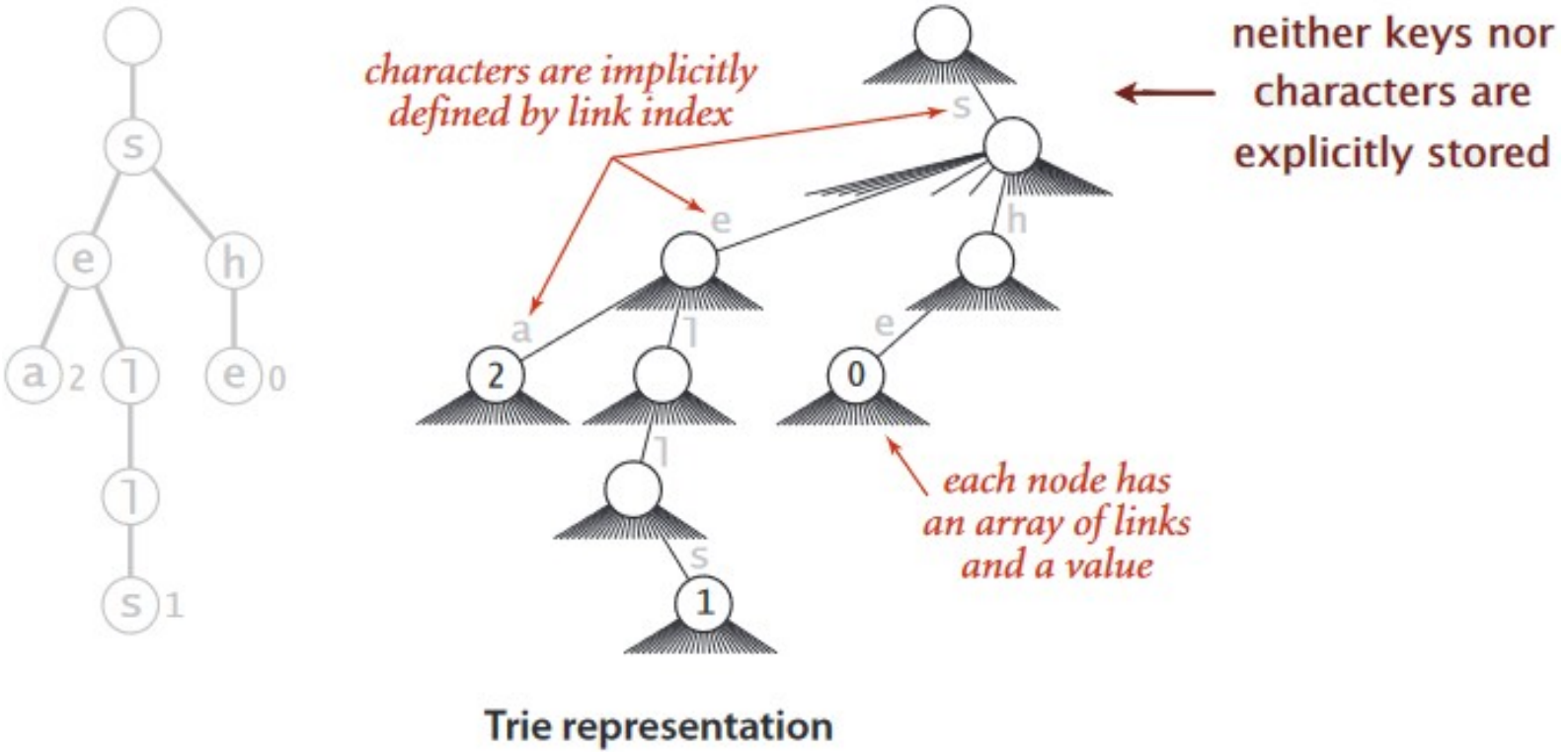
Tipos de Tries

- R-way
- DST – Digital Search Tree
- Suffix Tree
- Patricia Tree
- DAWG – Directed Acyclic Word Graph
- TST – Ternary Search Tree

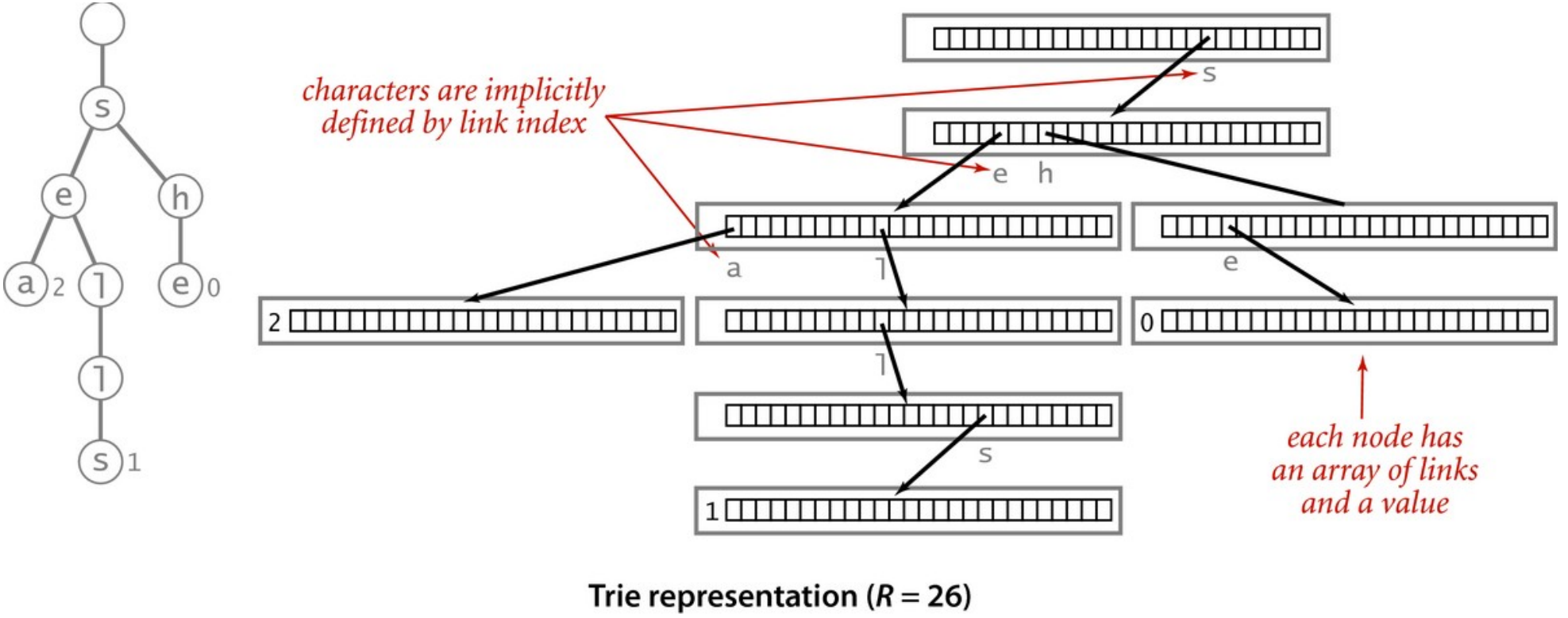
Implementação como R-Way Trie

- A implementação padrão e mais simples das árvores trie é conhecida como **multiway trie** ou **r-way trie** (considerando um alfabeto do **R** símbolos).
- Nesta implementação, cada nó contém apontadores para todos os **R** valores do alfabeto, mais um campo que sinaliza se aquele é um nó terminal, isto é, se ele forma uma chave completa ou não.
- Os caracteres não são armazenados explicitamente, mas implicitamente através dos links.

Implementação como R-Way Trie



Implementação como R-Way Trie



Trie representation ($R = 26$)

Implementação como R-Way Trie

```
struct TrieNode
{
    struct TrieNode *children[ALPHABET_SIZE];
    bool isEndOfWord;
};
```

Complexidade

- O tempo de execução das operações de busca, inserção e remoção é **O(key_length)**
  - não depende do número de elementos da árvore, mas sim do comprimento da chave
- A complexidade de espaço é de

  O(ALPHABET_SIZE * key_length * N), onde N o número de chaves na trie
  - O maior problema é que há um grande desperdício de espaço (símbolos do alfabeto que não possuem nenhum filho).
  - Há implementações mais eficientes em termos de espaço, como as tries compactas, TST – Ternary Search Tree, etc.

Aplicações de Tries

- Dicionários, como aqueles utilizados para operações de autocompletar e corretor ortográfico
- Programas para compreender linguagem natural
- Ordenação: construir uma trie com os filhos de cada nodo em ordem. Percorrer a árvore em pré-order (tipo de radix sort)
- Substituir tabelas hash e BST em algumas situações
- Compressão de dados
- Tabelas de roteamento para endereços IP
- Armazenar e consultar documentos XML
- Tabela de símbolos de compiladores
- Pesquisas em texto de grandes dimensões
- Construção de índices de documentos
- Expressões regulares (padrões de pesquisa)