

Busca Digital (Trie e Árvore Patrícia)

Estrutura de Dados II
Jairo Francisco de Souza

Introdução

- No problema de busca, é suposto que existe um conjunto de chaves $S = \{s_1, \dots, s_n\}$ e um valor x correspondente a uma chave que se deseja localizar em S .
- Nos métodos vistos até agora, tentavam estruturar S de alguma forma conveniente e, através de comparações de x com s_i de S , tentar localizar x em S .
- Nesses métodos, cada chave s_i , bem como a chave desejada x , é tratada como um único elemento indivisível.

Introdução

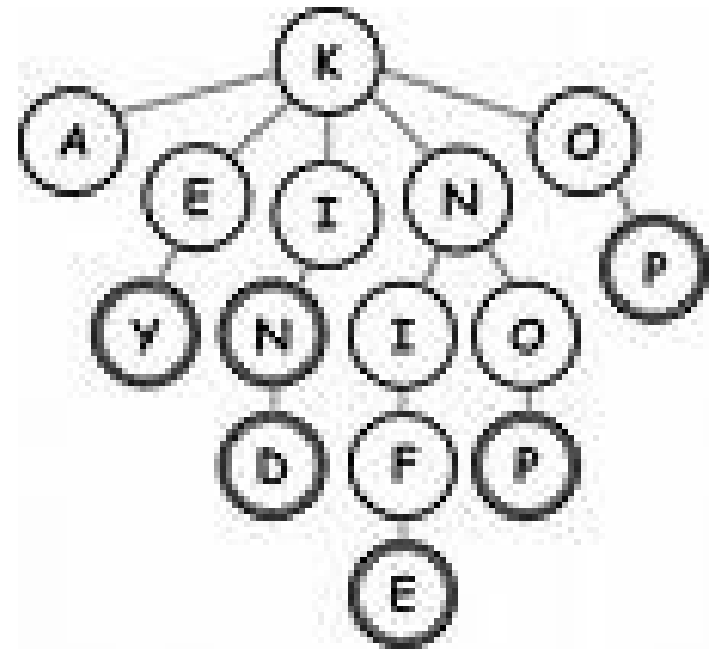
- Porém, nem sempre as chaves serão do mesmo tamanho e podem exceder o espaço definido para elas.
- Suponha que se deseje armazenar um texto literário para, em seguida, tentar localizar frases nesse texto.
- Neste caso, o conjunto S de chaves corresponderia às frases armazenadas e cada s_i à uma frase passível de ser buscada.
- Neste cenário, a busca digital é a mais apropriada.

Busca digital

- A diferença entre a busca digital e a busca estudada até agora é que a chave não é tratada como um elemento indivisível.
- Isto é, assume-se que cada chave é constituída de um conjunto de caracteres ou dígitos definidos em um alfabeto apropriado.
- Em vez de se comparar a chave procurada com as chaves do conjunto armazenado, a comparação é efetuada, individualmente, entre os dígitos que compõem as chaves, dígito a dígito.
- 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

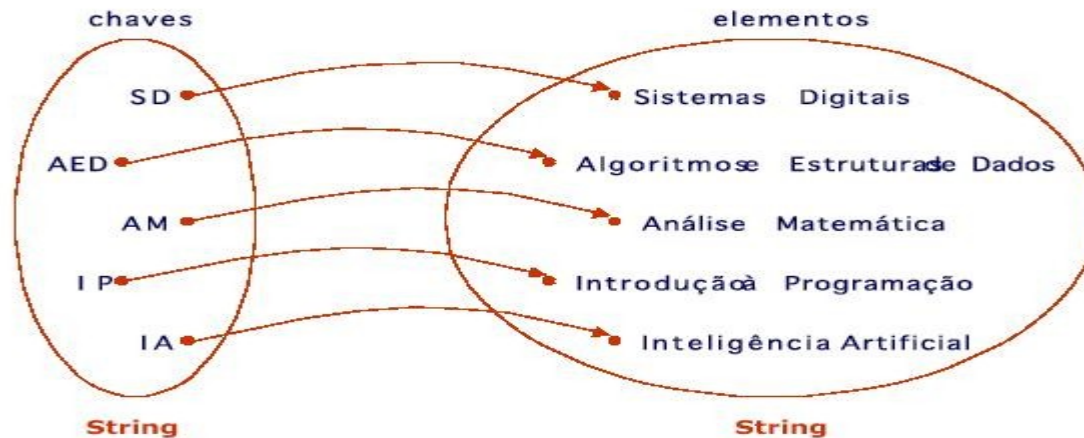
Introdução

- TRIE vem de RETRIEVAL – RECUPERAÇÃO
 - Pronúncia: TRI ou TRAI
- É um tipo de árvore de busca.
- Idéia geral: usar partes das CHAVES como caminho busca
- Origem: anos 60 por Edward Fredkin



CHAVES: Características

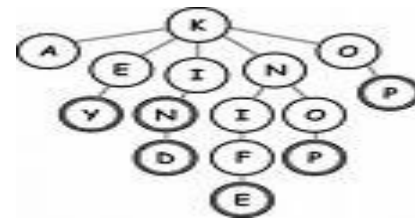
- Cada **chave** formada por **palavras** sobre um **alfabeto**
- Palavras com tamanho **variável** e **ilimitado**
- Em geral associam-se **chaves** a **elementos** ou **registros**, como na tabela Hash



CHAVES: Características

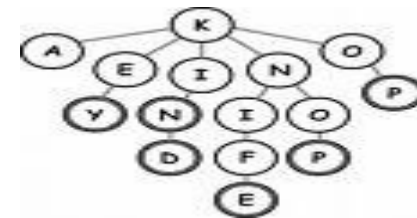
- Cada **chave** é formada a partir de **alfabeto** de símbolos
- Exemplos de **alfabetos**:
 $\{0,1\}$, $\{A, B, C, D, E, \dots, Z\}$, $\{0,1,2,3,4,5, \dots, 9\}$
- Exemplos de **chaves**:
ABABBBBABABA 19034717 Maria
010101010000000000101000000001010
- Chaves parcialmente **partilhadas** entre os elementos

Trie: A estrutura



- Árvore **ordenada** e **n-ária**
- Chaves em geral **caracteres**
- Ao contrário da árvore de busca binária **nenhum nó** armazena a chave
- Chave determinada pela **posição** na árvore

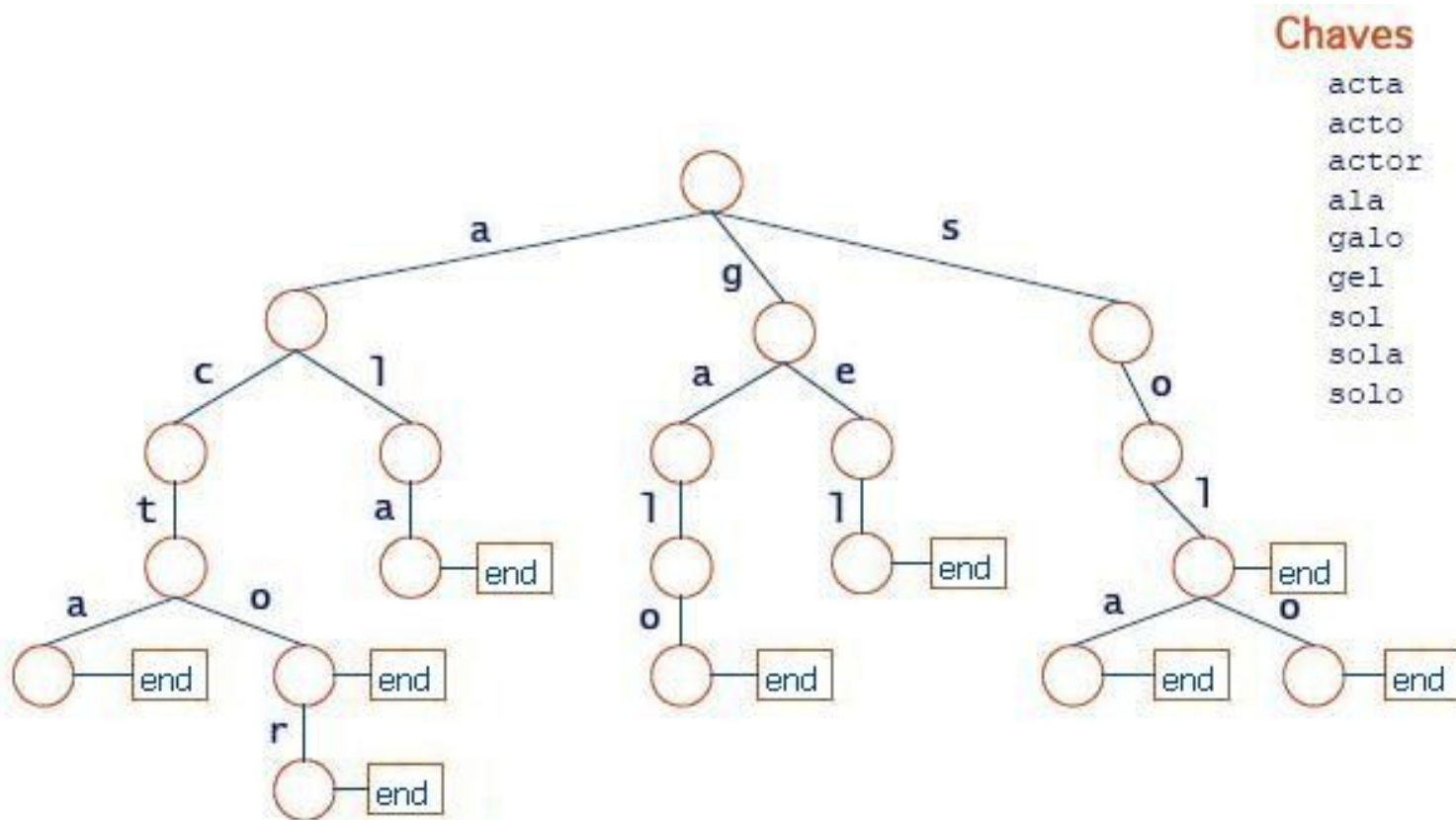
Trie: A estrutura



- Descendentes de mesmo nó com mesmo prefixo
- Raiz: cadeia vazia
- Valores ou elementos associados a folhas ou a alguns nós internos de interesse
- O caminho da raiz para qualquer outro nó é um prefixo de uma string

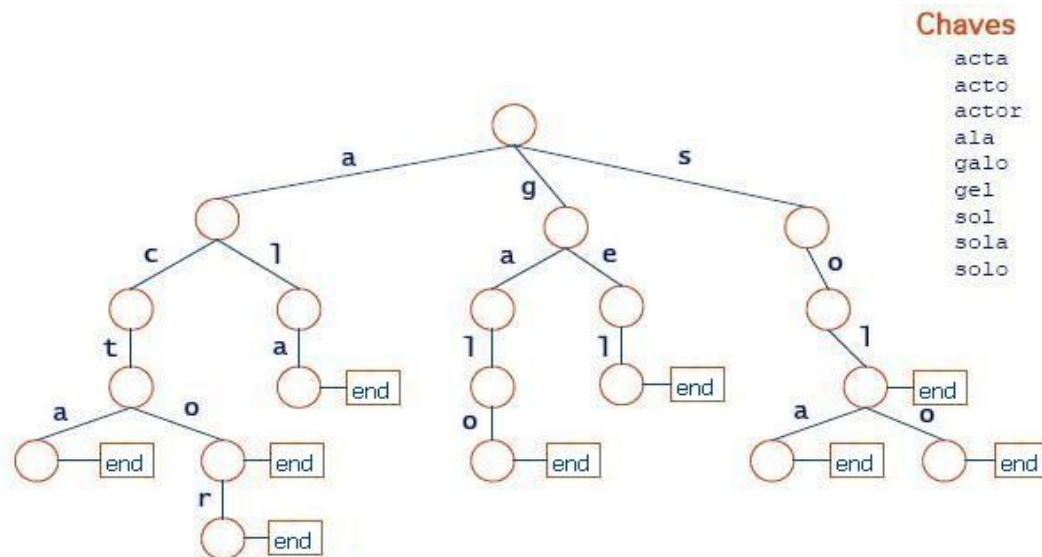
Trie: A estrutura

- Árvore ordenada e n-ária



Trie: A estrutura

- O **grau** corresponde ao tamanho do **alfabeto**
- A **trie** pode ser vista como um **autômato finito**
- Cada **nível** percorrido corresponde a avançar um **elemento** na chave



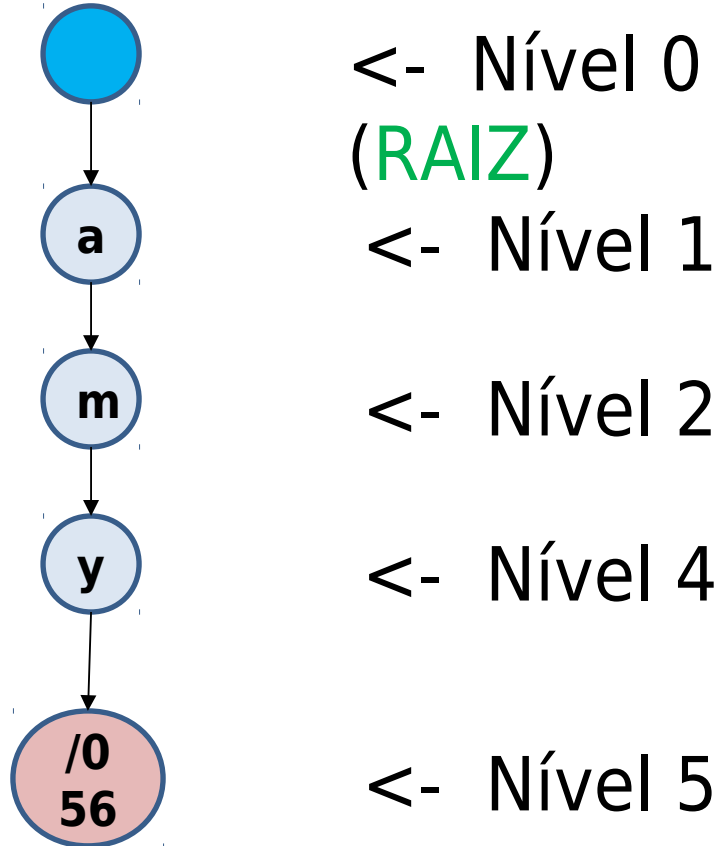
Montando uma árvore TRIE

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

Estes são pares que queremos **colocar** na árvore TRIE

Montando uma árvore TRIE

- amy 56



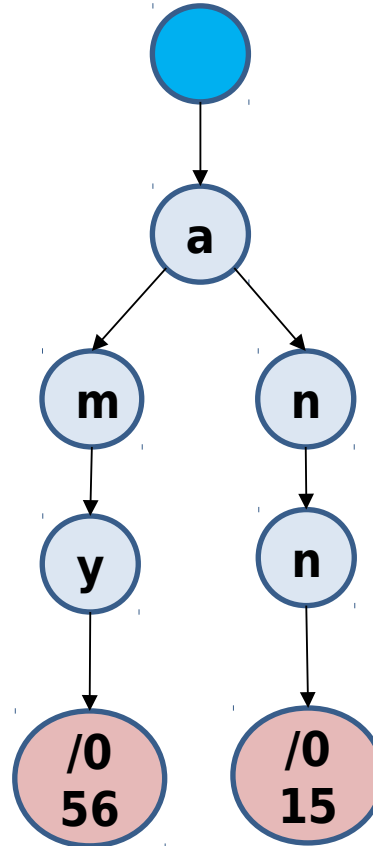
Montando uma árvore TRIE

- INSIRA

ann 15

Montando uma árvore TRIE

- ann 15



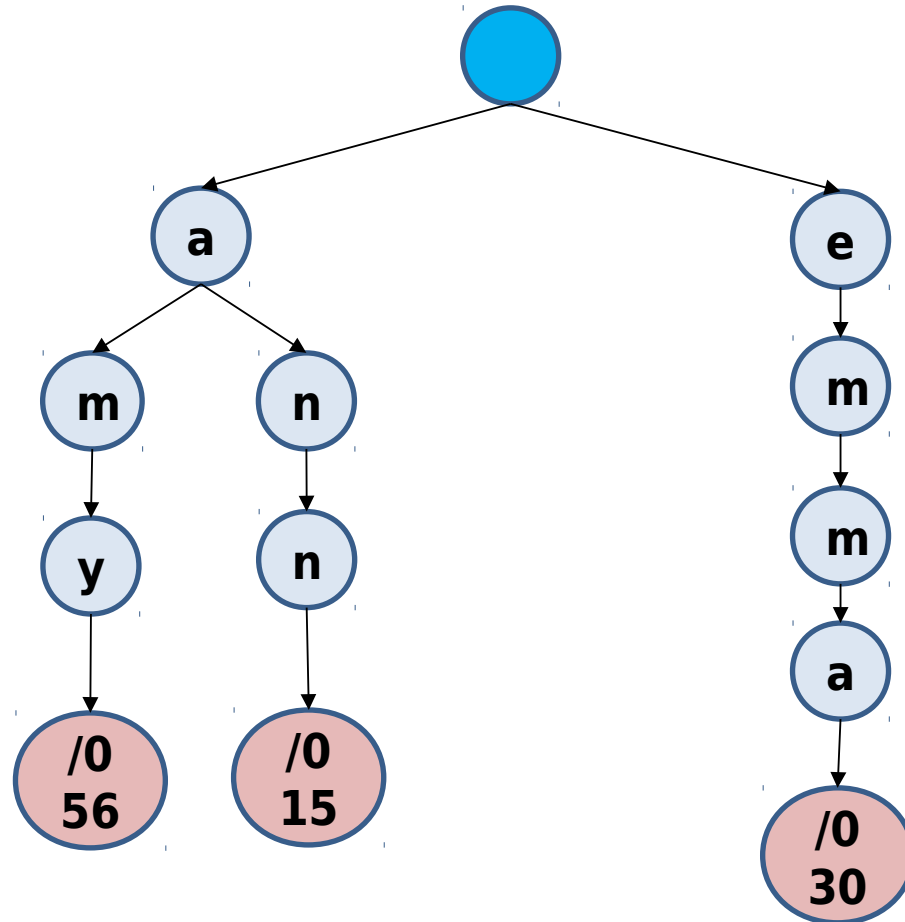
Montando uma árvore TRIE

- INSIRA

emma 30

Montando uma árvore TRIE

- emma 30



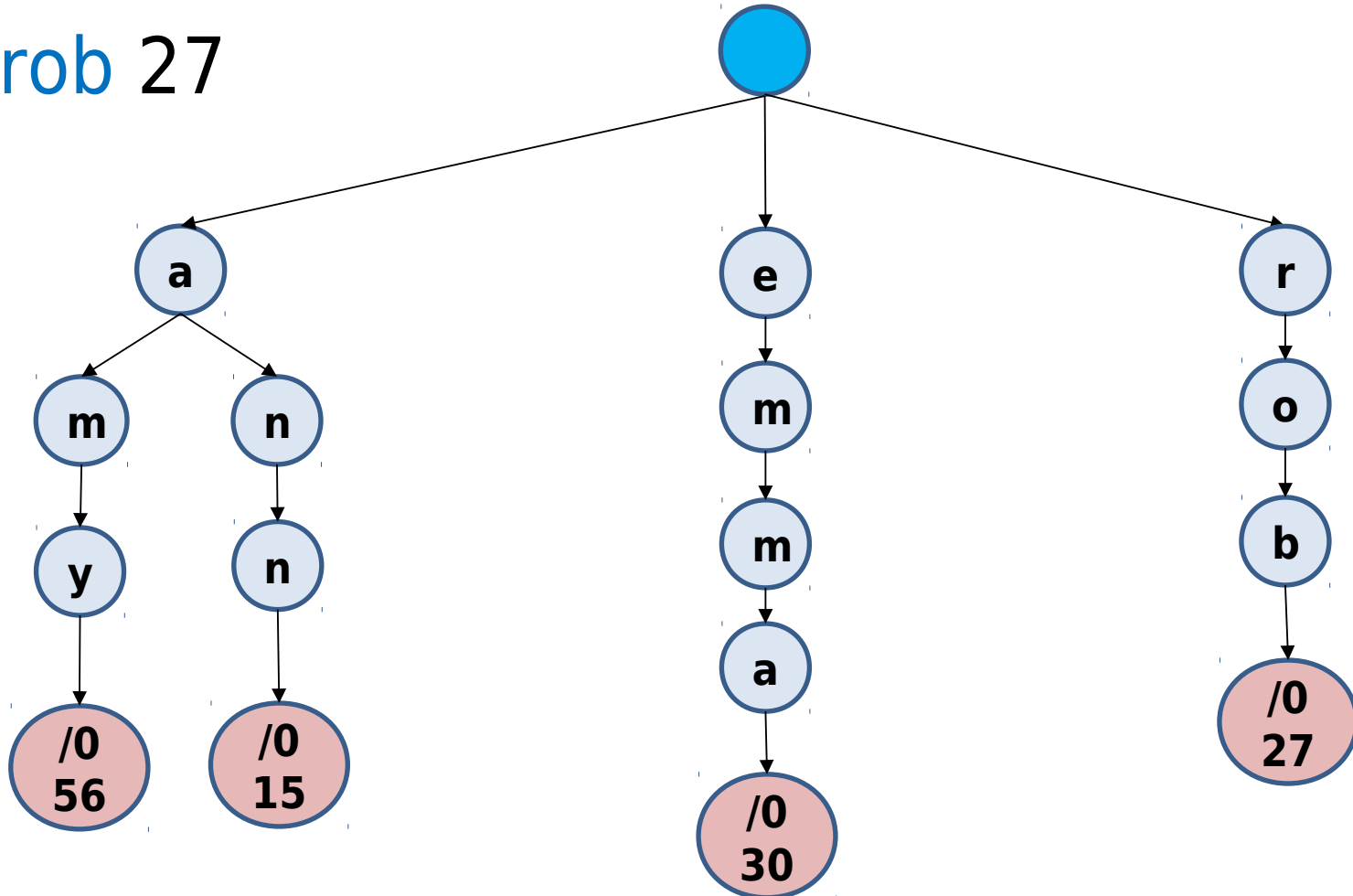
Montando uma árvore TRIE

- INSIRA

rob 27

Montando uma árvore TRIE

- rob 27



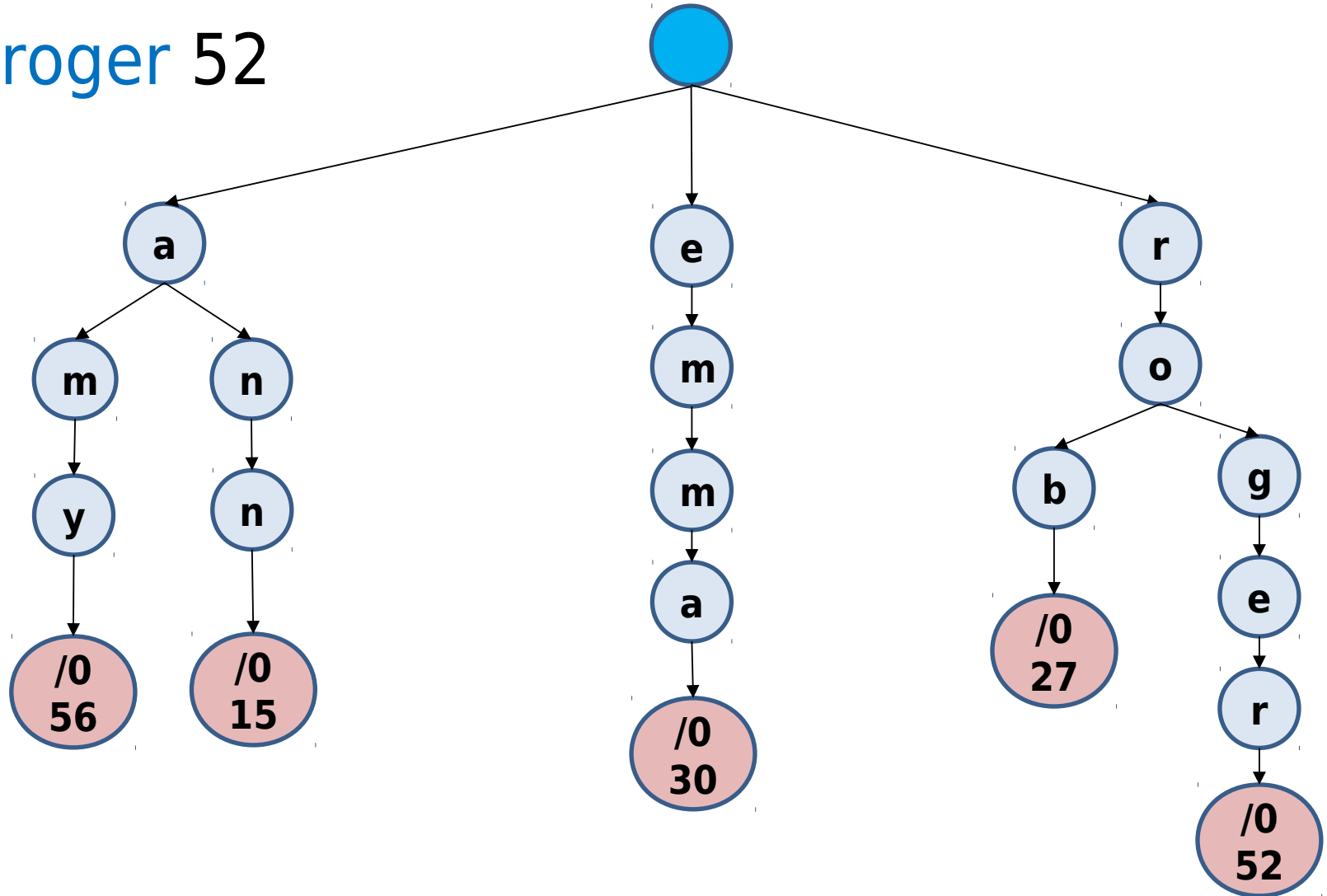
Montando uma árvore TRIE

- INSIRA

roger 52

Montando uma árvore TRIE

- roger 52



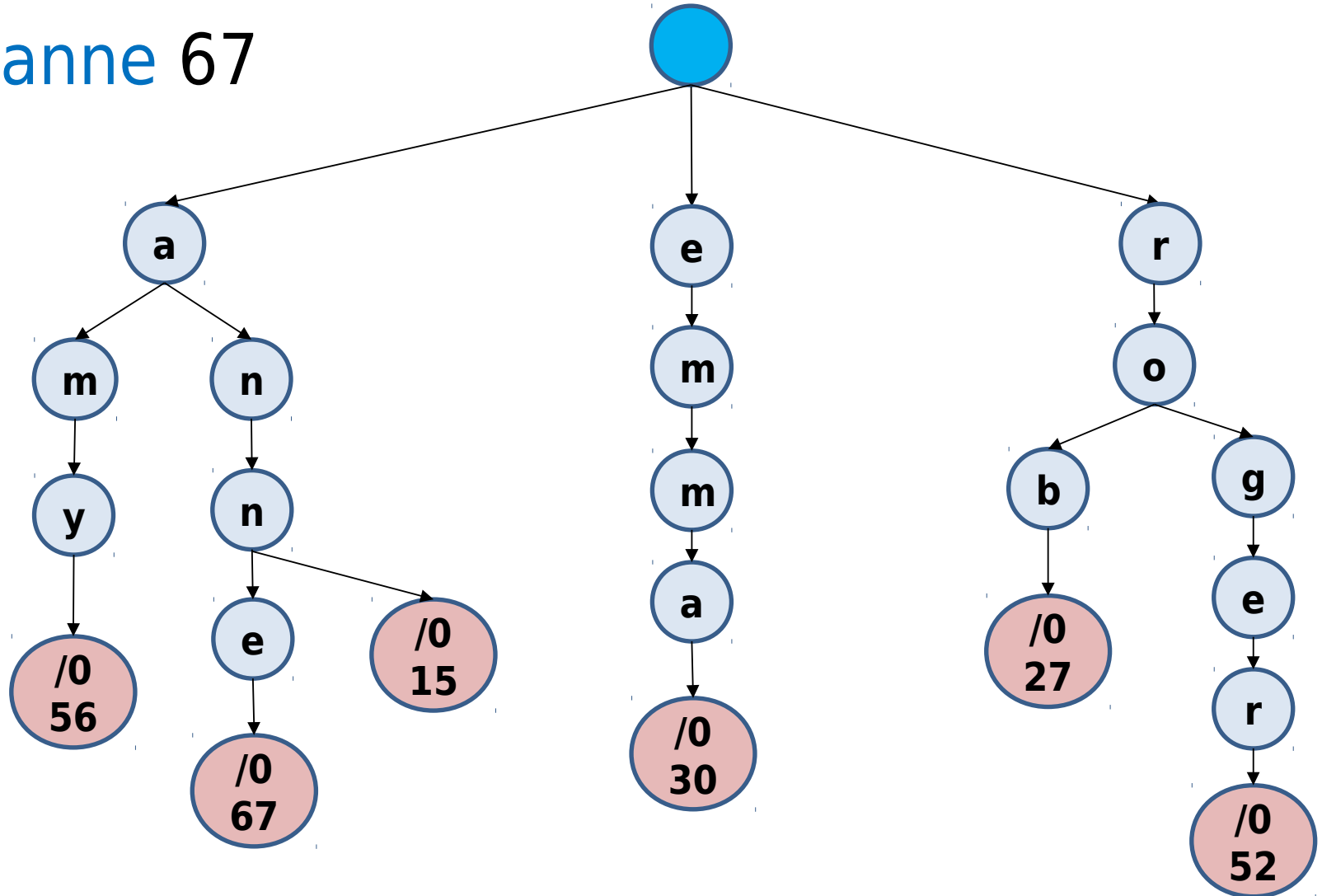
Montando uma árvore TRIE

- INSIRA

anne 67

Montando uma árvore TRIE

- **anne** 67



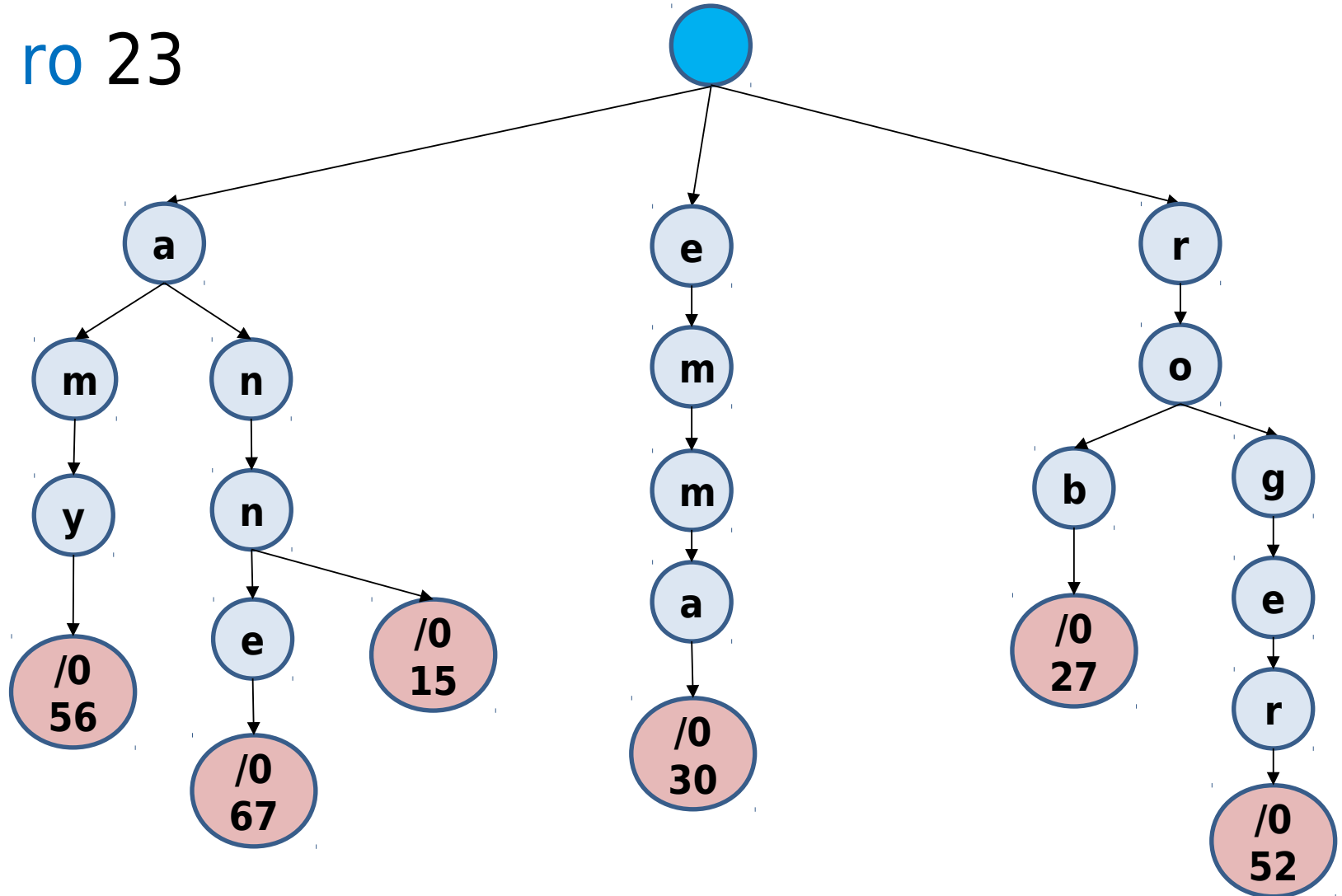
Montando uma árvore TRIE

- INSIRA

ro 23

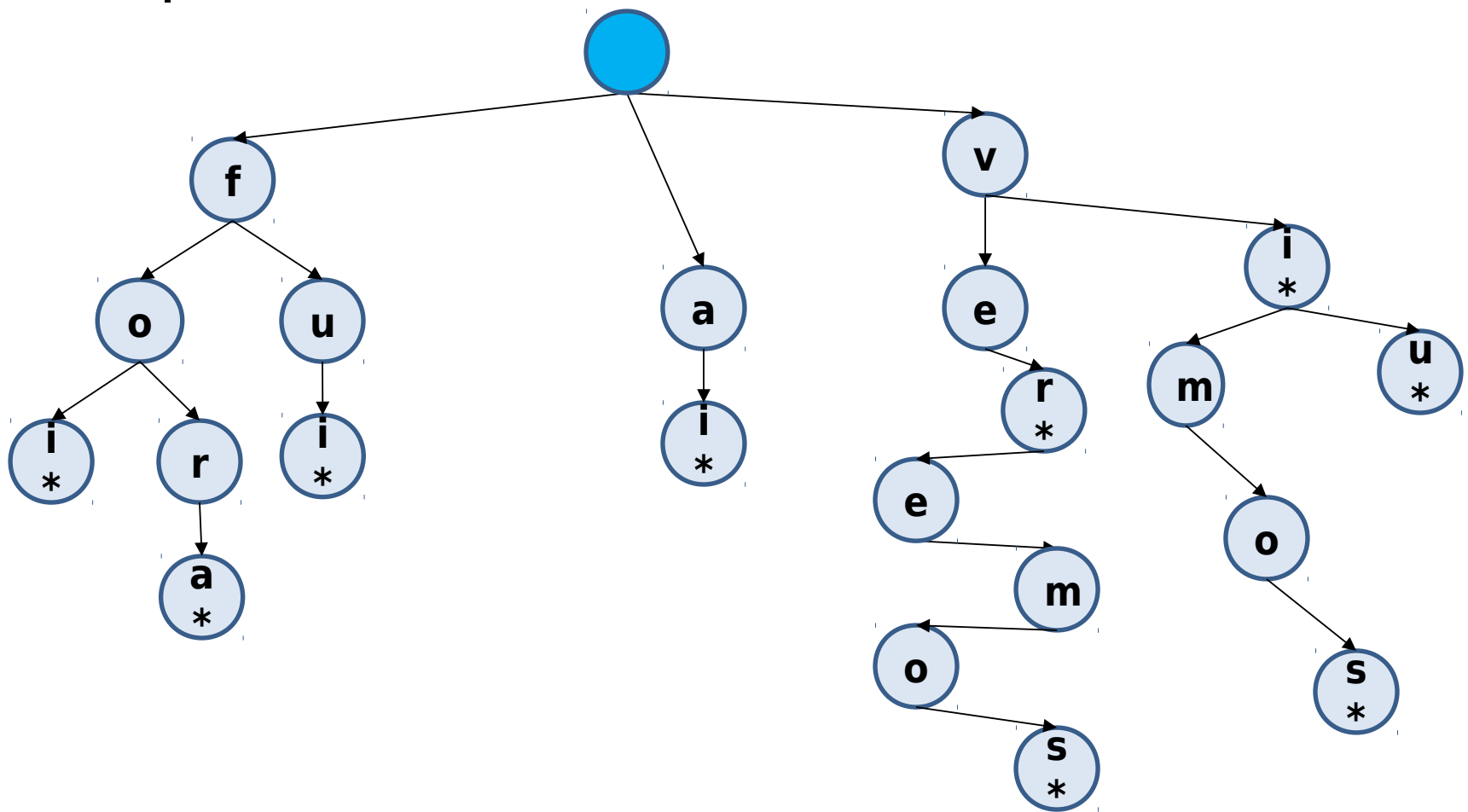
Montando uma árvore TRIE

- ro 23



EXERCÍCIO

- Quais chaves/palavras estão representadas nesta trie?

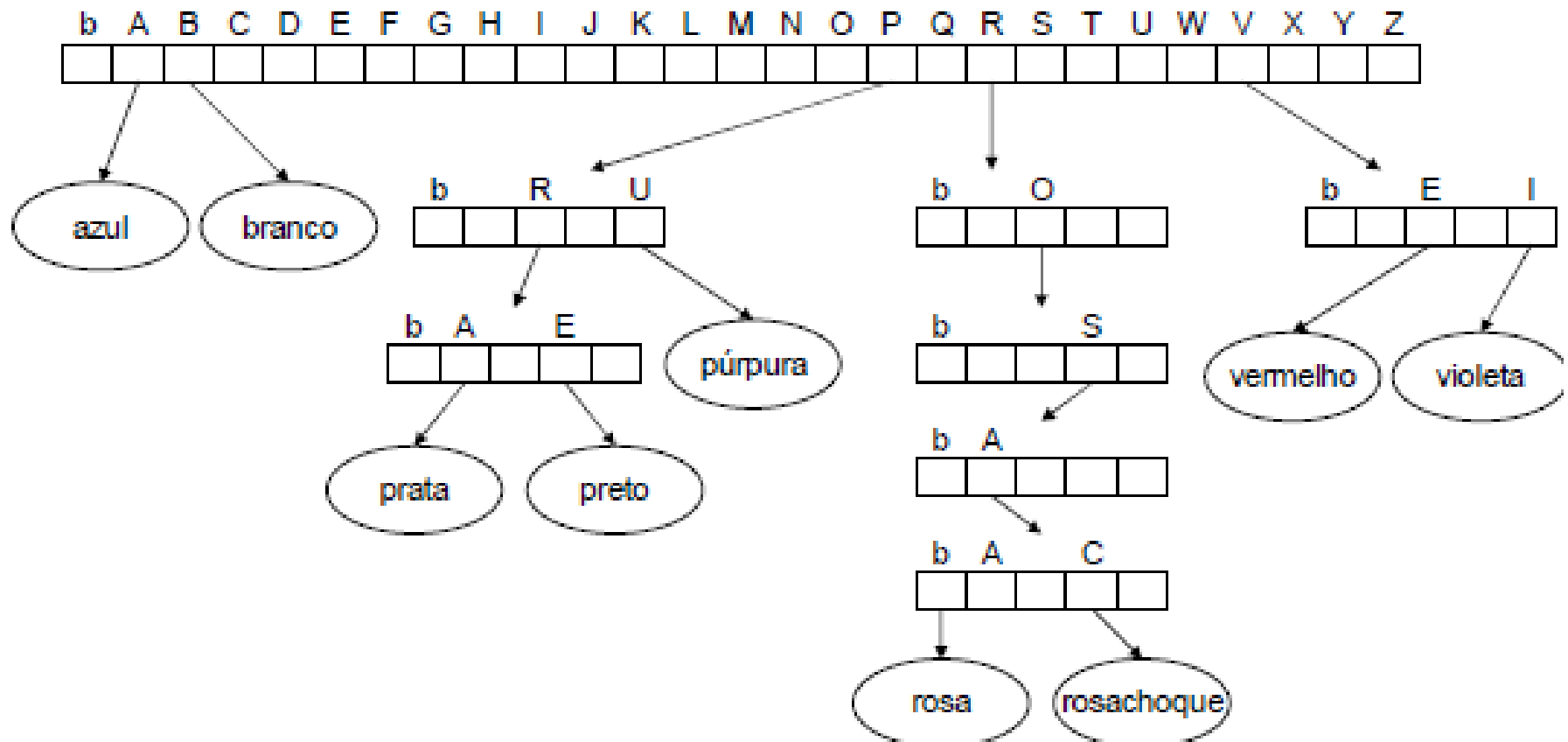


Implementando uma TRIE

- Implementação mais simples: R-WAY
 - A árvore contém dois tipos de nós: nó de desvio e nó de informação.
 - Cada nó de desvio contém todos os valores do alfabeto mais 1 símbolo especial para determinar uma chave.
 - Há desperdício de espaço.
- Considere uma trie para armazenar chaves do alfabeto {a, b, c, d, ..., z}
 - Ou seja, 27 letras

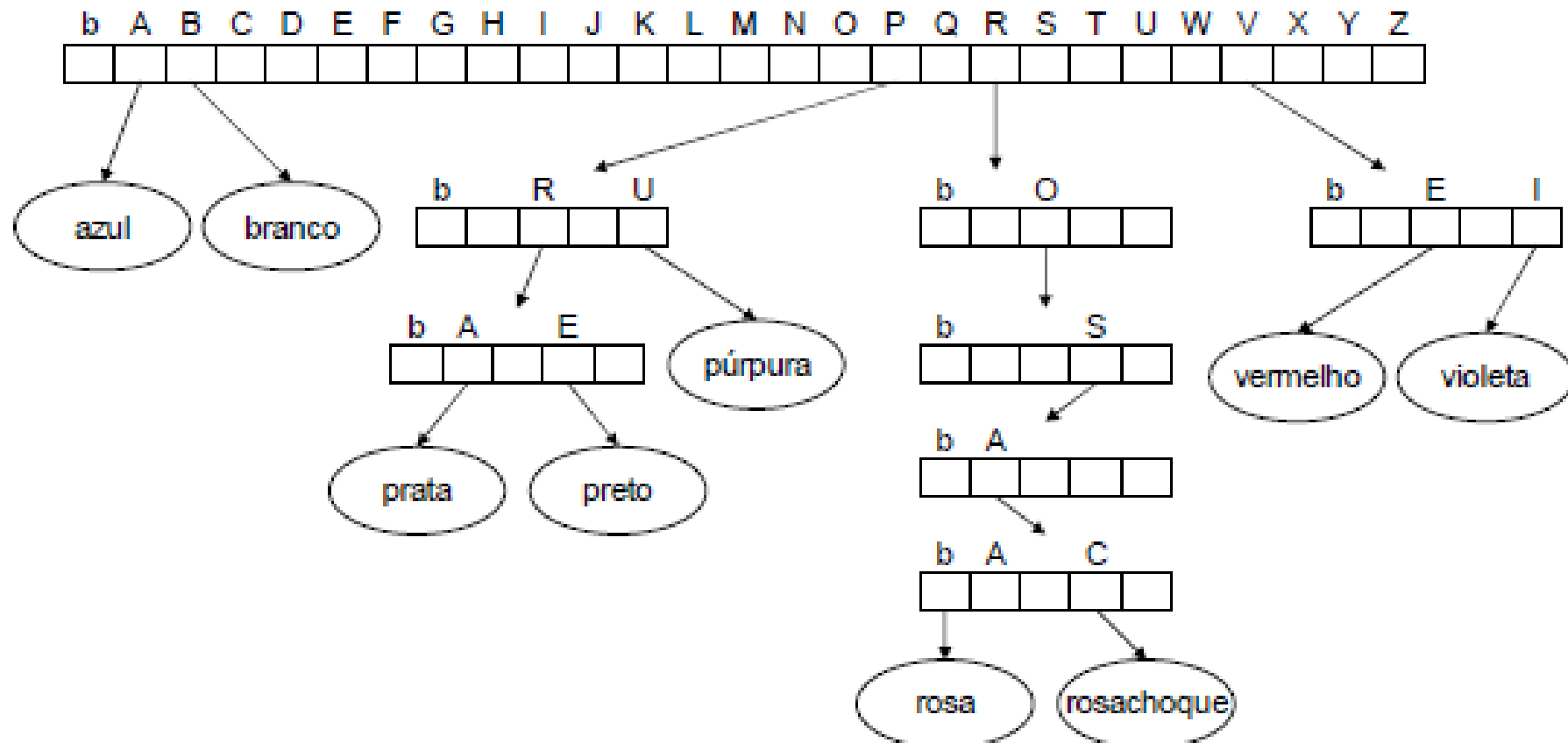
Implementando uma TRIE

- A árvore seguinte contém dois tipos de nós: nó de desvio e nó de informação.



Implementando uma TRIE

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



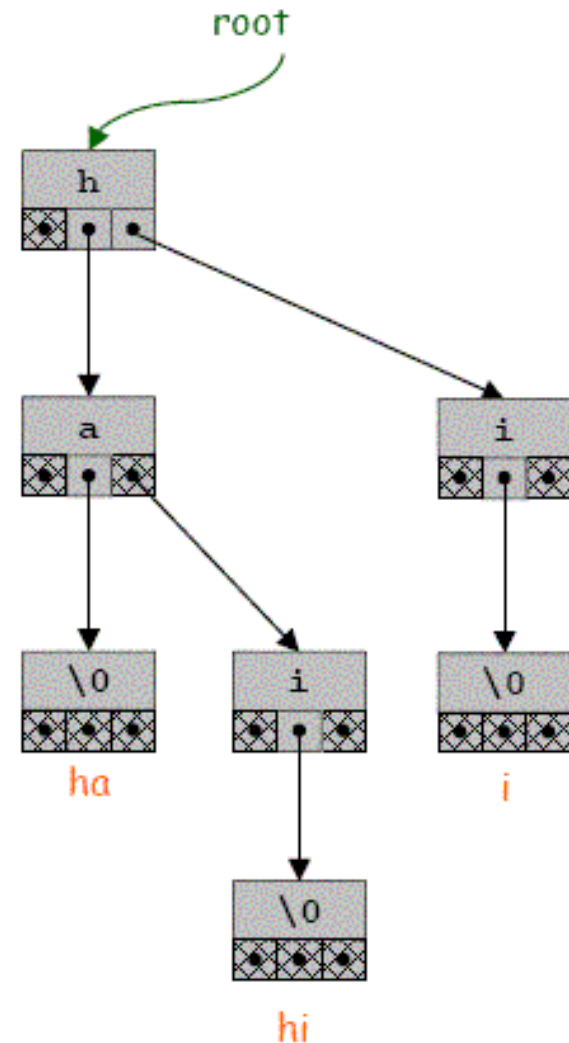
Implementando uma TRIE

TST- Ternary Search Tree

Cada nó aloca três ponteiros

Centro: caractere seguinte
Filhos da **esquerda** e **direita**:
caracteres alternativos

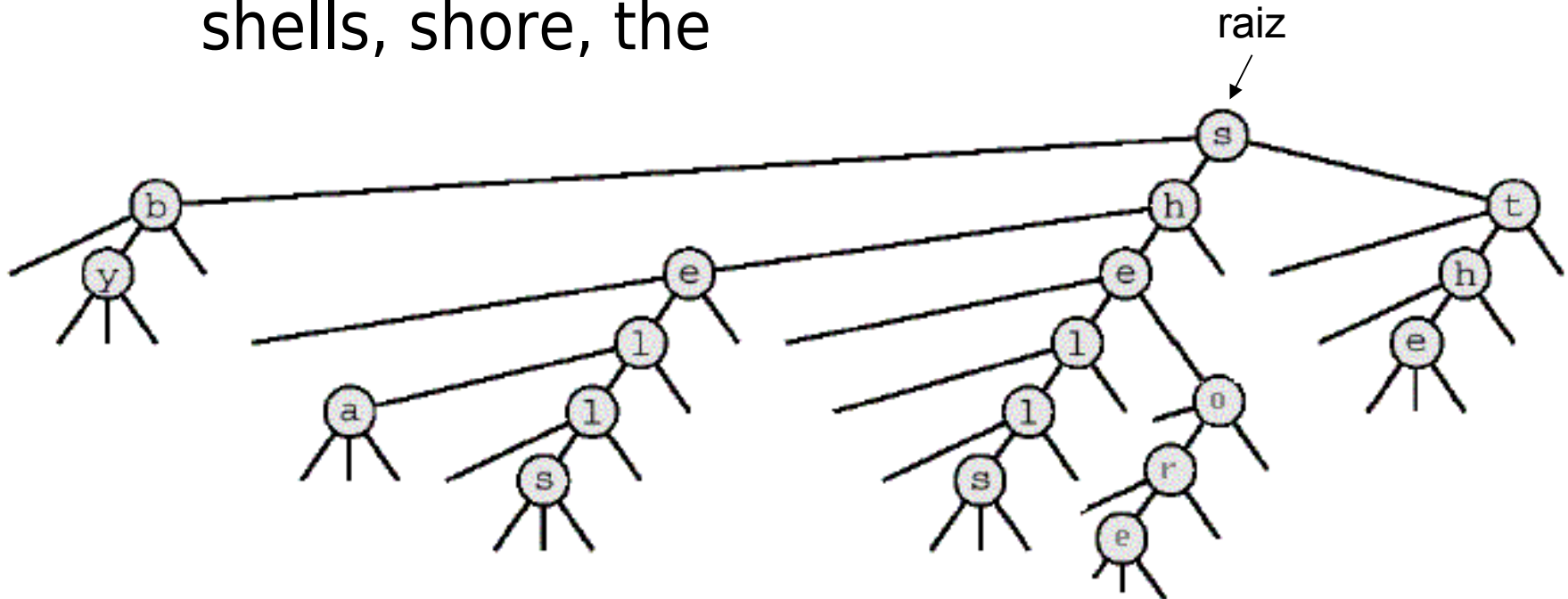
Tem **desempenho melhor** no que
se refere a espaço.



Implementando uma TRIE

TST- Ternary Search Tree

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



Operações em TRIES

- **INSERÇÃO**

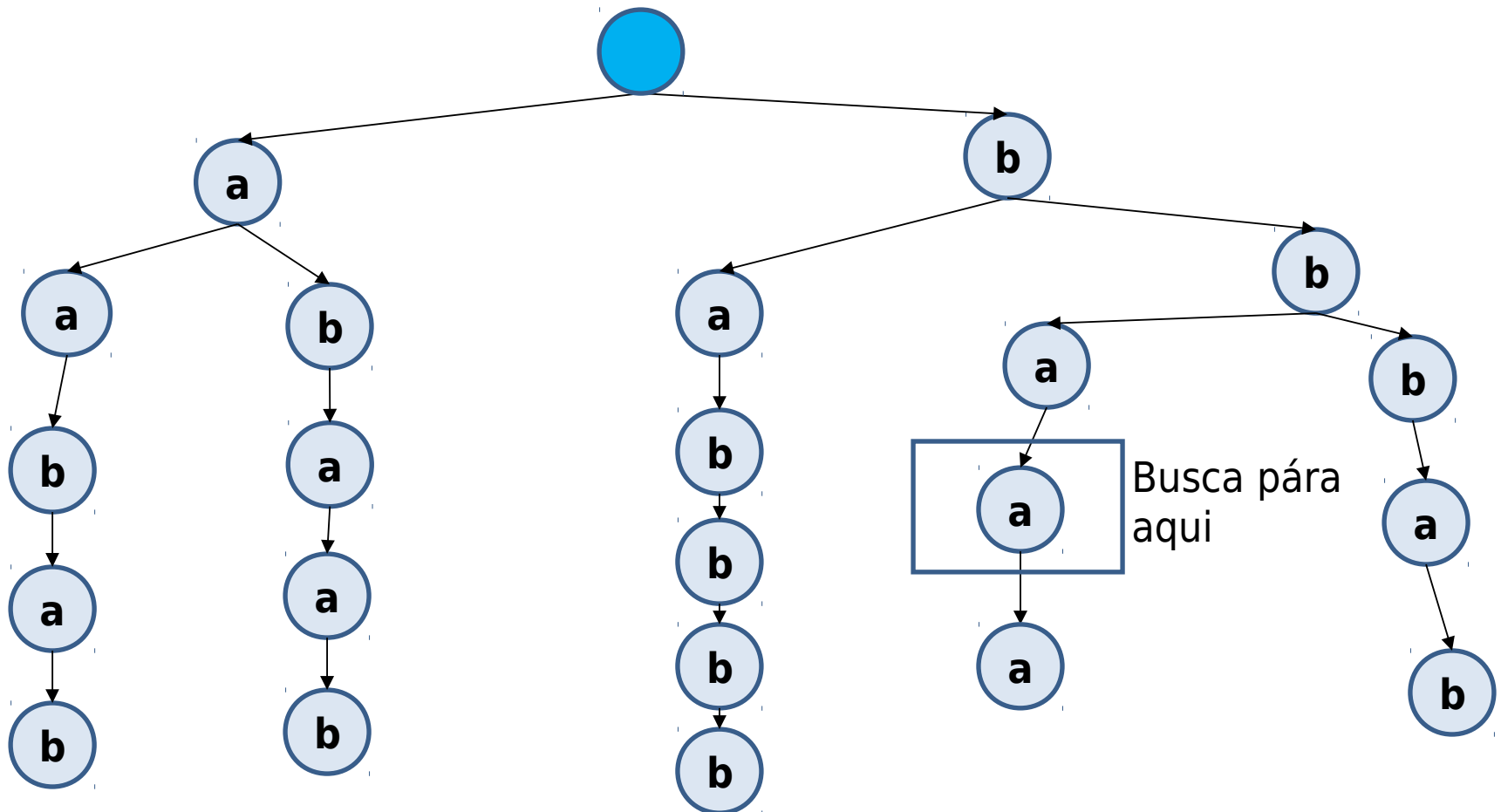
Faz-se uma **busca** pela palavra a ser inserida.
Se ela já existir na TRIE nada é feito.

Caso contrário, é recuperado o nó até onde acontece a maior **substring** da palavra a ser inserida.

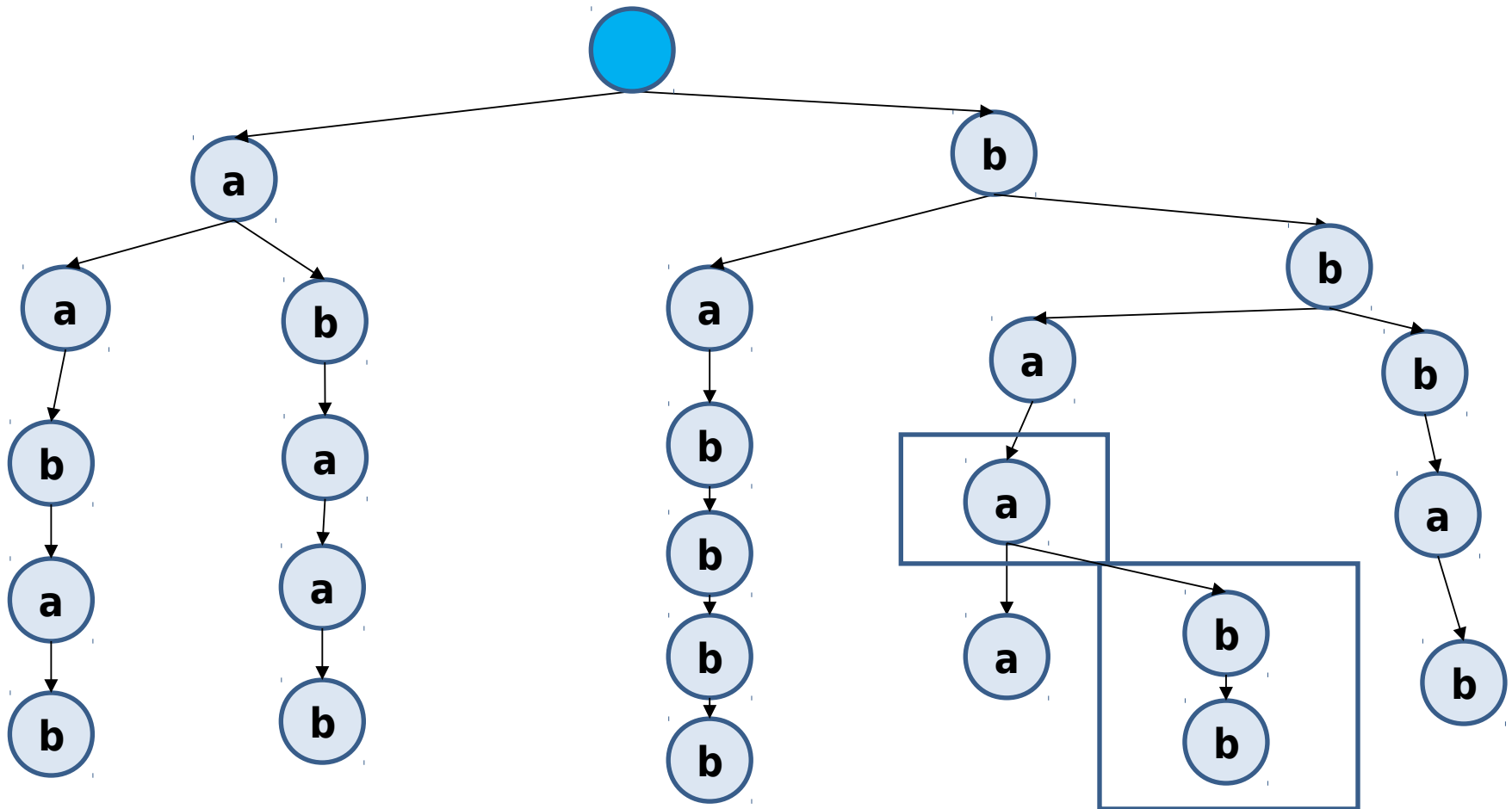
O **restante** dos seus caracteres são adicionados na TRIE a partir daquele nó

Operações em TRIES

Inserção : **bbaabb**



Inserção : bbaabb



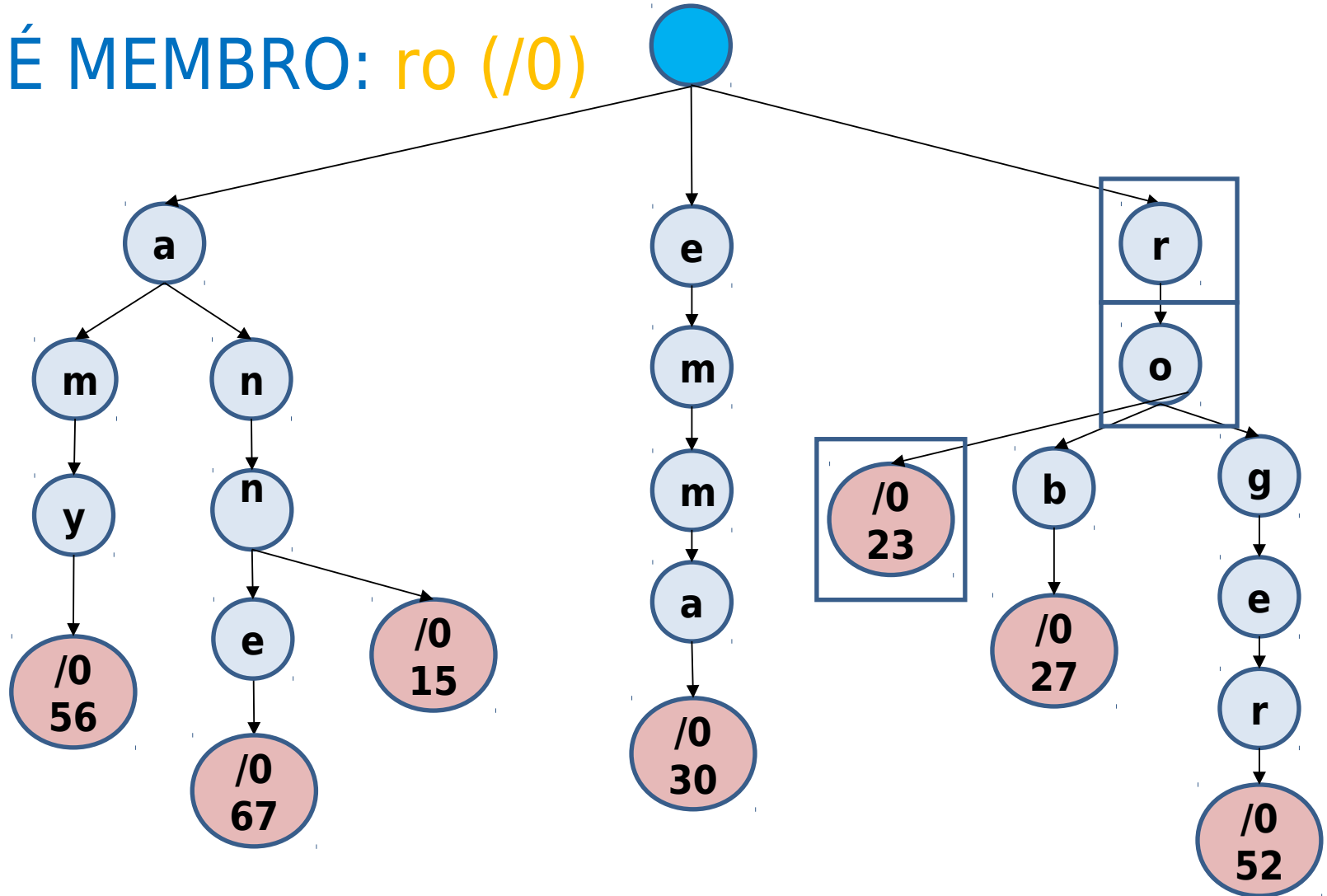
Operações em TRIES

- É MEMBRO

1. Busca no nível superior o nodo que confere com o primeiro caractere (corrente) da chave
2. Se nenhum nó confere, retorna **FALSO**
Senão
3. Se o caractere que confere é \0
Retorna **Verdadeiro**
Senão
4. Move para a **subTRIE** que confere com esse caractere
5. Avança para o próximo caractere na chave
6. Vá para **passo 1**

Operações em TRIES

- É MEMBRO: ro (/0)



Operações em TRIES

- REMOÇÃO

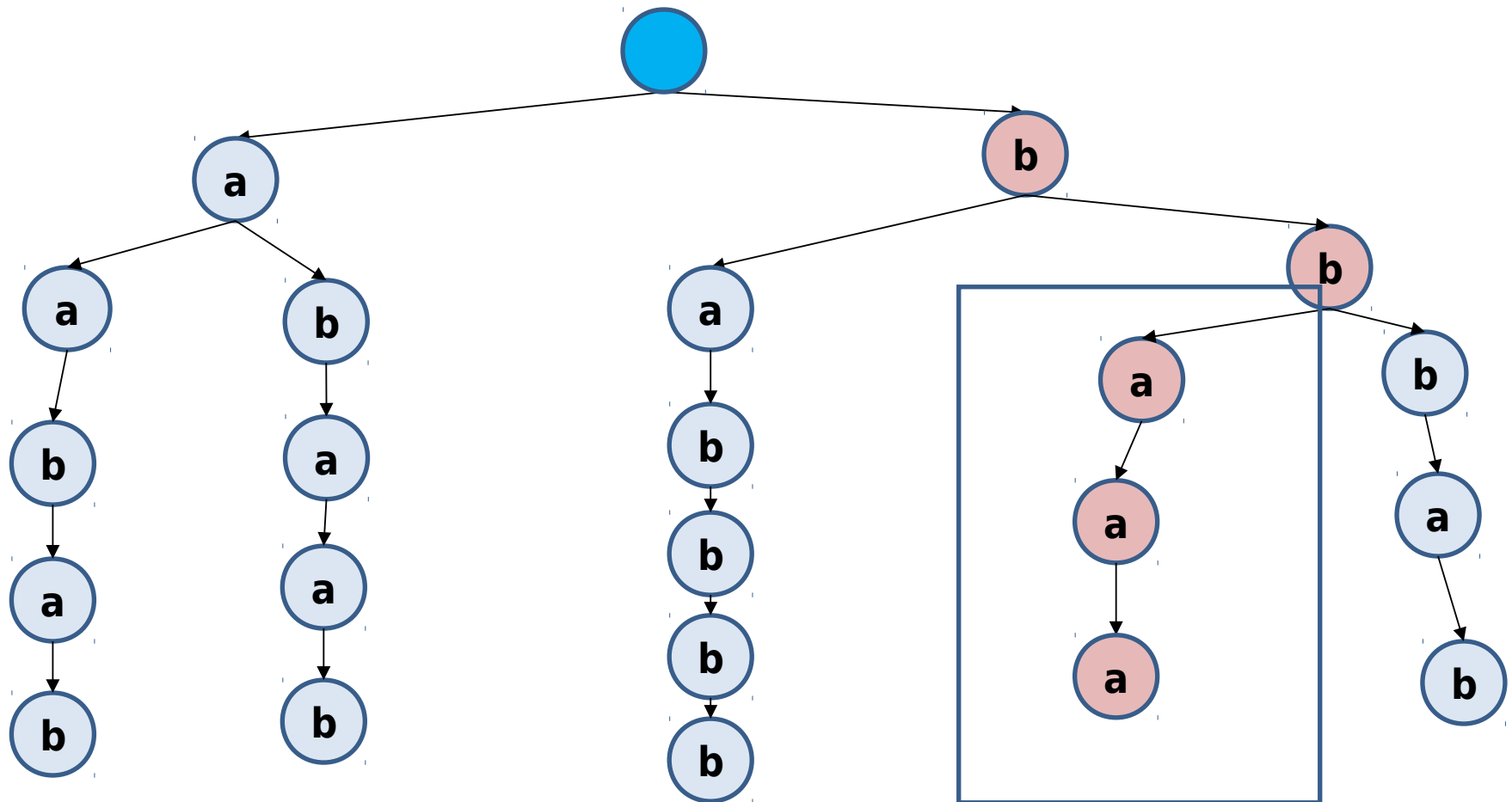
Busca-se o **nó** que representa o final da palavra a ser removida.

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

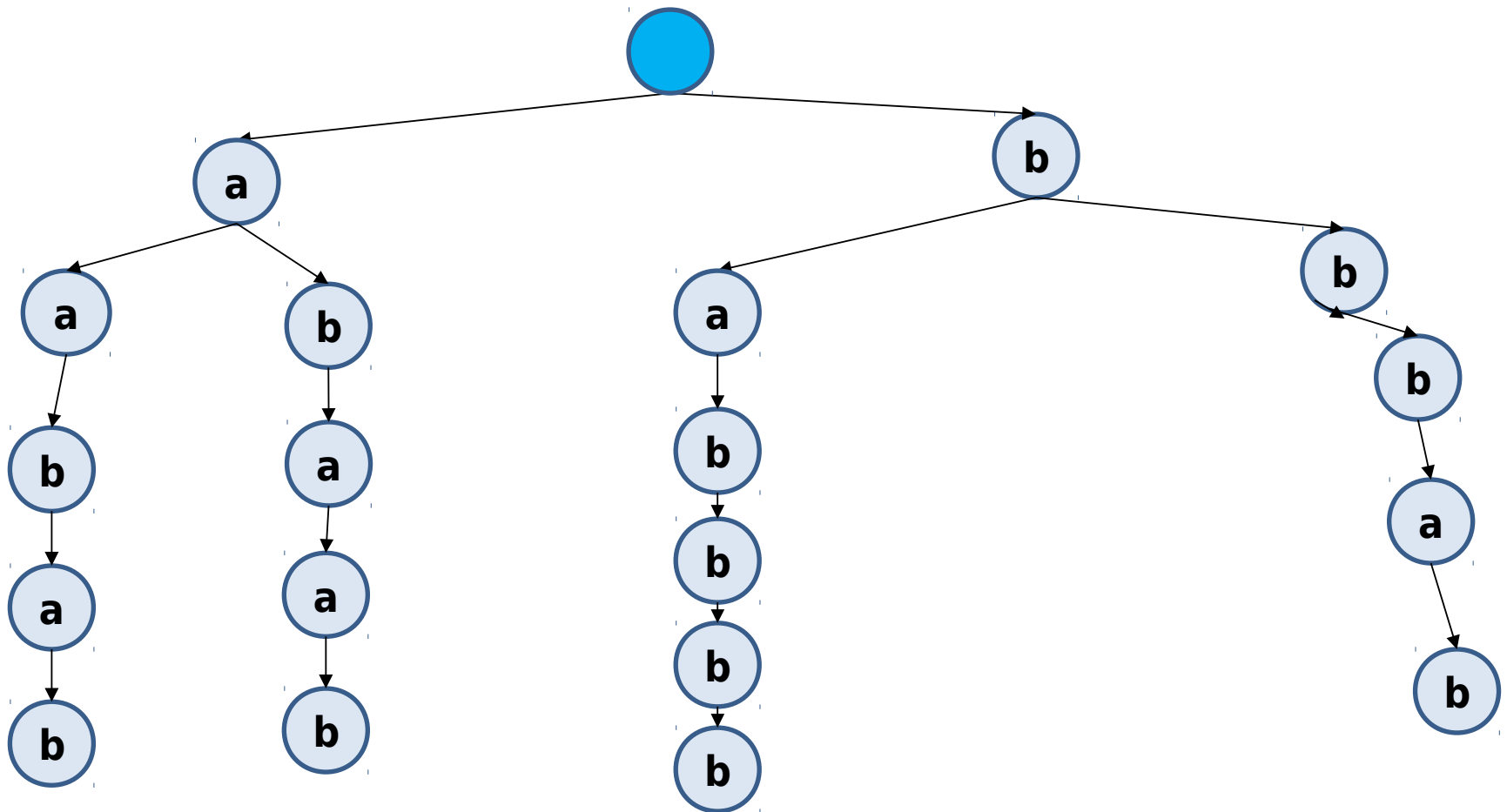
Operações em TRIES

Remoção : **bb**aaaa



Operações em TRIES

Remoção : **bb**aaaa



Operações em TRIES

- COMPLEXIDADE

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

Complexidade: $O(AK)$

A = tamanho do **alfabeto**

K = tamanho da **chave**

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 de **PREFIXO** e a **PATRÍCIA**

Árvore Digital Binária

- Árvore digital binária é simplesmente o caso binário da árvore digital, 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.

Árvore Digital Binária: Exemplo

- Chaves:

00 → *

0000

00010 \Rightarrow +

00011

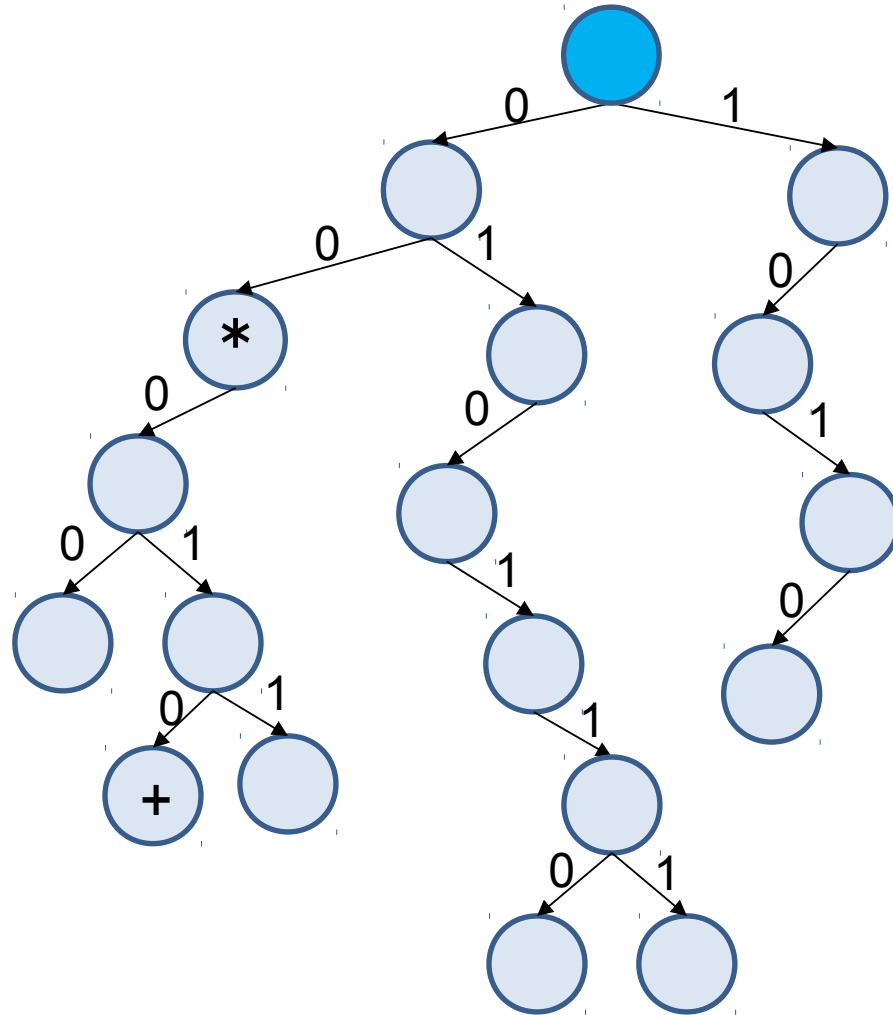
010110

010111

10

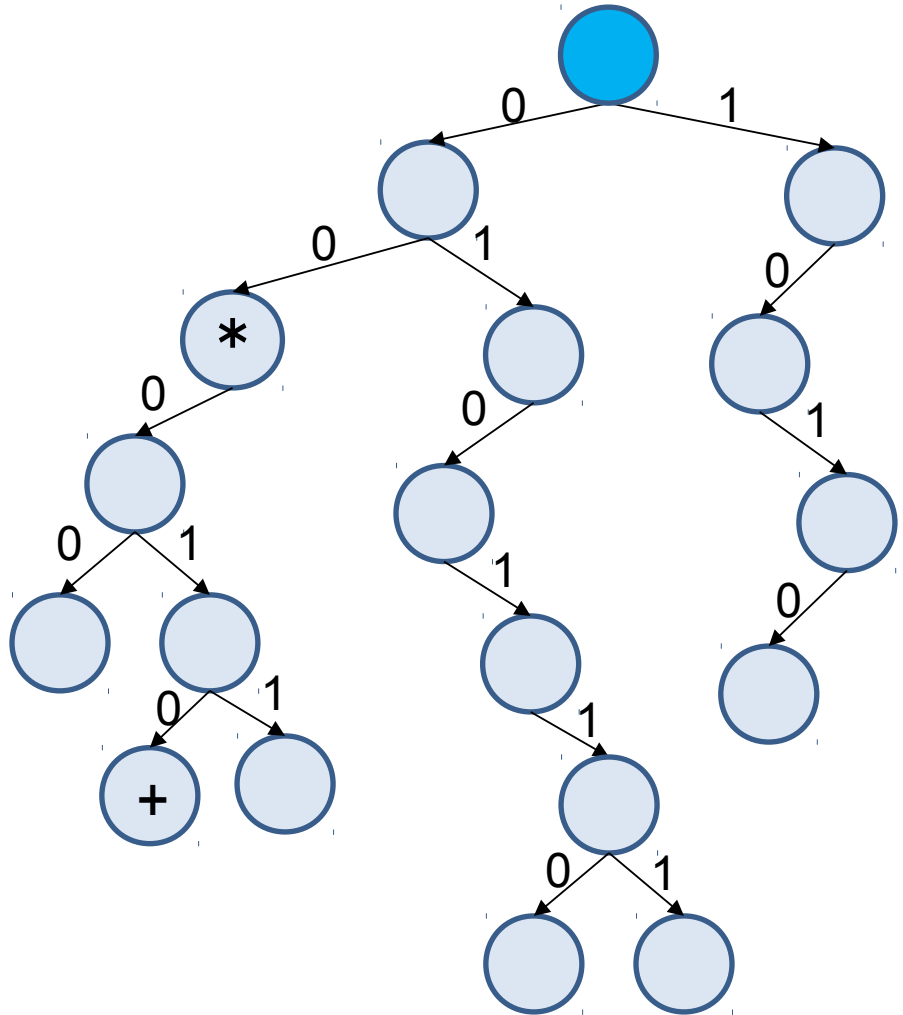
101

1010



Árvore Digital Binária

- 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



Árvore Binária de Prefixo

- 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.

Árvore Binária de Prefixo

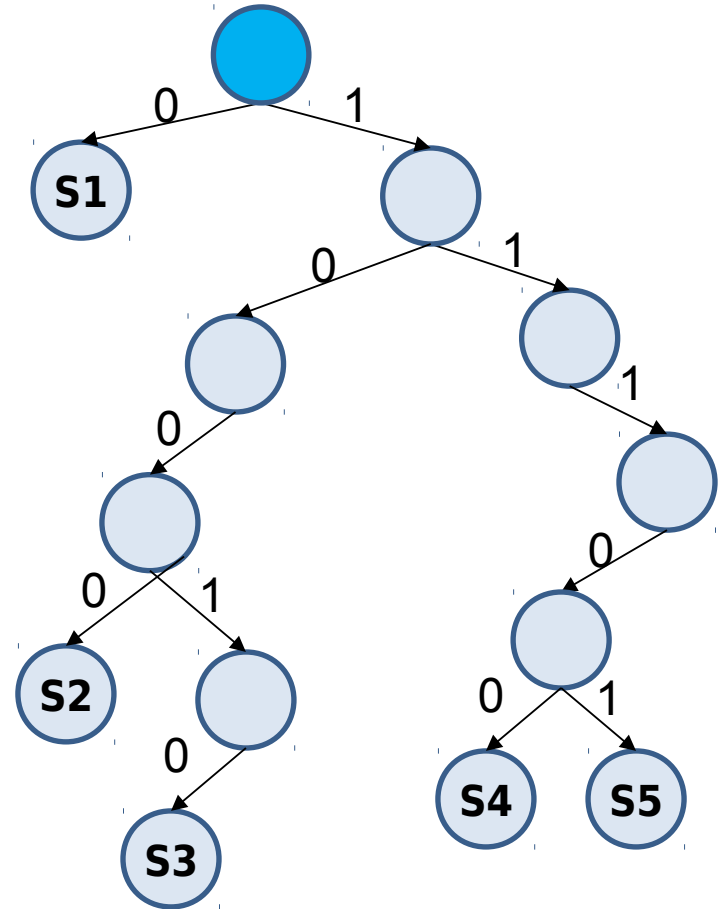
S1 = 0

S2 = 1000

S3 = 10010

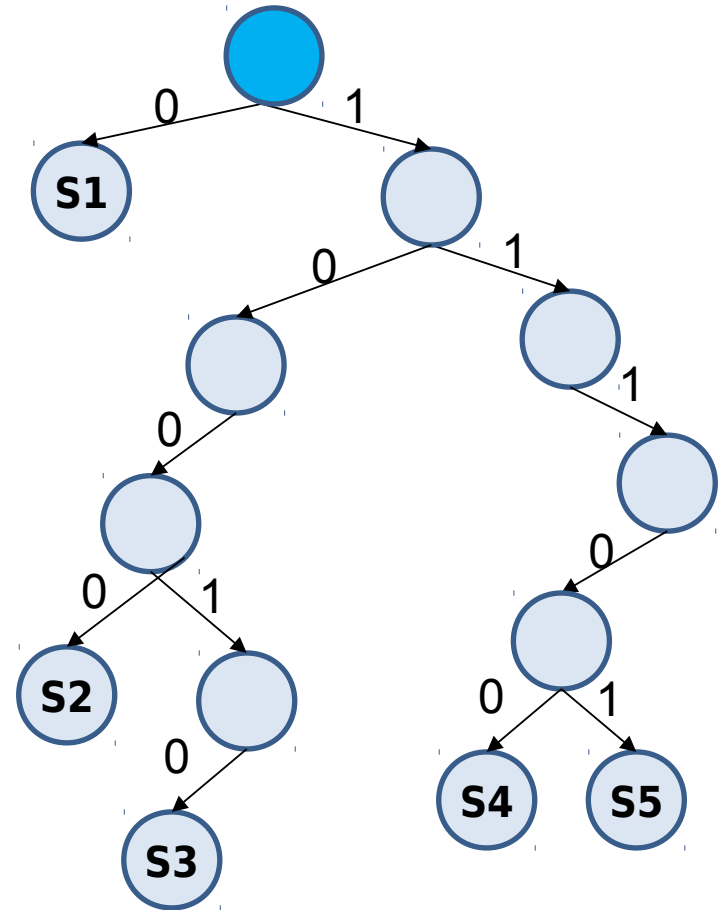
S4 = 11100

S5 = 11101



Árvore Binária de Prefixo

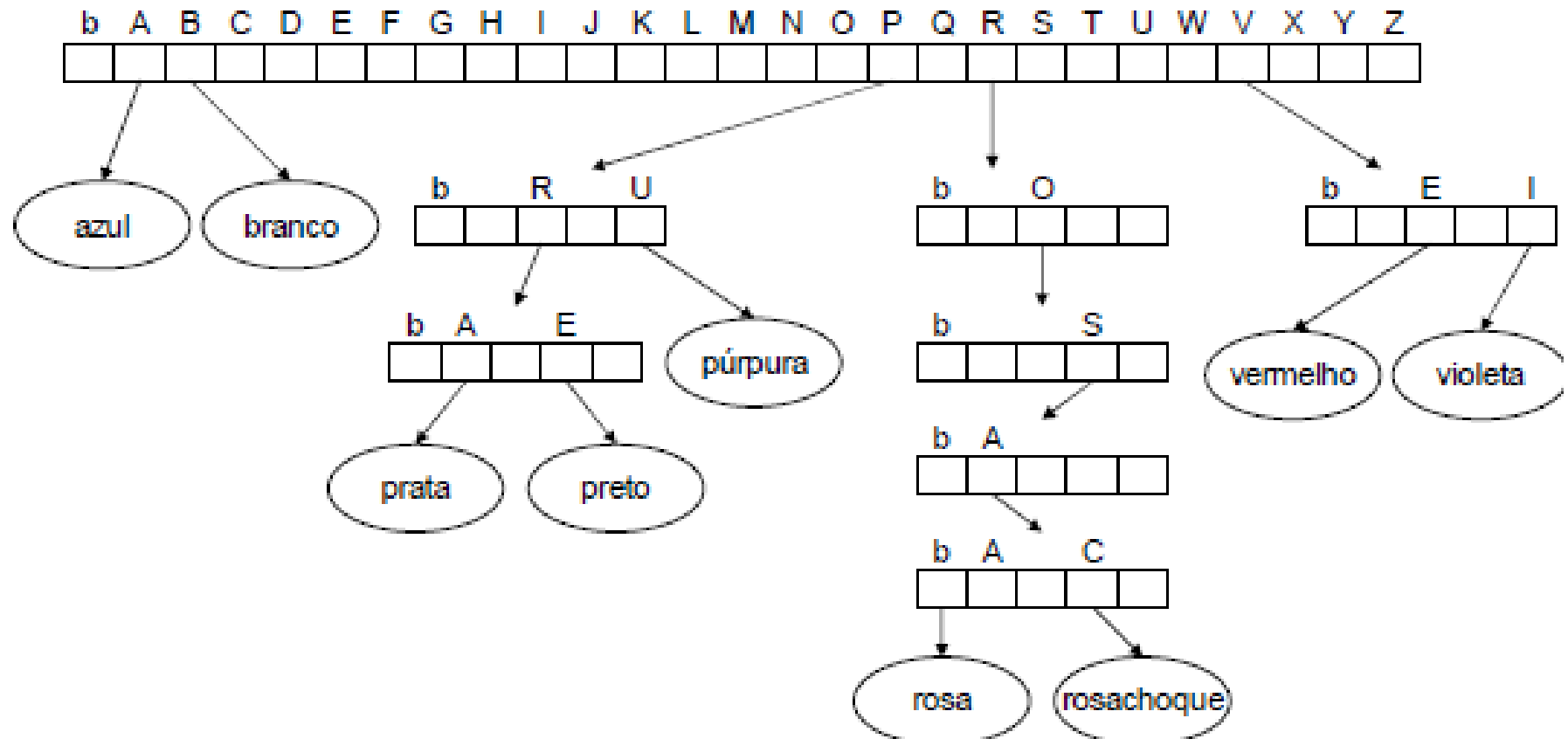
- Uma propriedade interessante da árvore binária de prefixo é que há uma correspondência entre o conjunto de chaves e o das folhas das árvores.
- Isto é, cada chave é unicamente representada por uma folha e a codificação binária dessa chave corresponde ao caminho da raiz até essa folha.



Outra alternativa

- Nem sempre poderá ser possível criar uma árvore de prefixo
 - Caso deseja-se incluir chaves rosa ou rosachoque, por exemplo
- Porém, podemos simplificar a estrutura evitando adicionar todo o caminho até o nó, caso o caminho até o nó seja o único a ser percorrido.
 - Por exemplo, ao inserir a chave “rosachoque”, não precisaríamos inserir o caminho “choque” caso só exista a chave “rosa”.
 - Essa solução é útil quando temos controle sobre as chaves que serão buscadas. Ou seja, não teríamos uma busca da chave “rosaclaro”!
 - Caso não se tenha controle das chaves que serão buscadas, é necessário verificar, ao final do caminho, se o valor da chave correspondente ao nó encontrado é **o mesmo** da chave pesquisada. Caso não seja, a chave não existe na árvore.

Outra alternativa



- Como implementar essa abordagem? Árvore Patrícia!

Árvore Patrícia

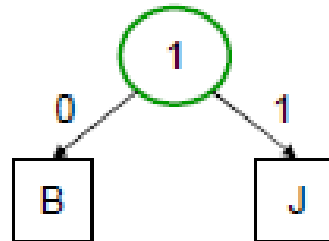
- Patricia é abreviatura de
 - Practical Algorithm To Retrieve Information Coded In Alphanumeric (Algoritmo Prático para Recuperar Informação Codificada em Alfanumérico)
- O algoritmo para construção da árvore Patricia é baseado no método de pesquisa digital, mas sem apresentar o inconveniente das tries.
 - É construída a partir da árvore binária de prefixo.
- 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 subárvore seguir

Inserção $X = B$



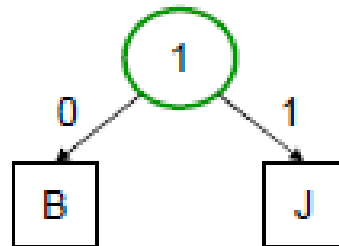
B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

Inserção $X = J$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

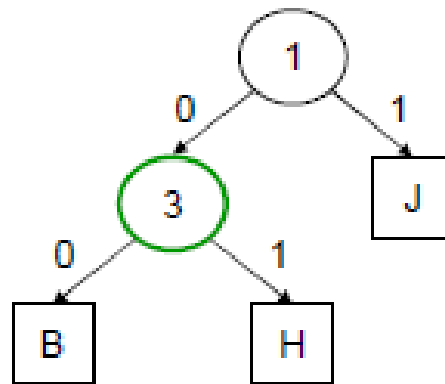
Inserção $X = H$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

B e H seguem o padrão
0xxxxx.

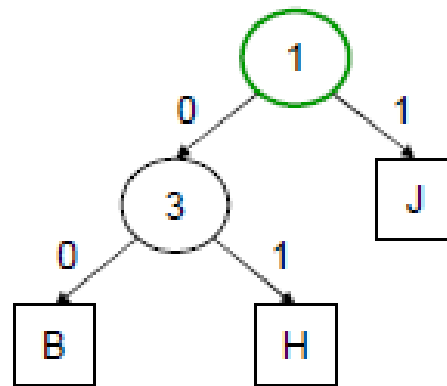
Inserção $X = H$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

B e H seguem o padrão
0xxxxx.
O 3o. bit diferencia B de H

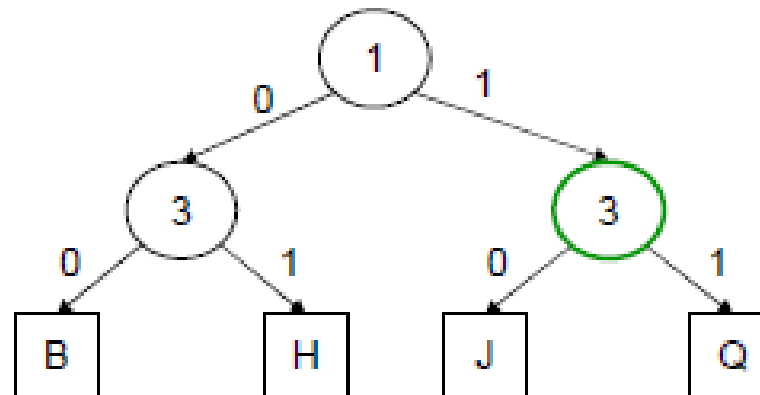
Inserção $X = Q$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

J e Q seguem o padrão
1xxxxx.

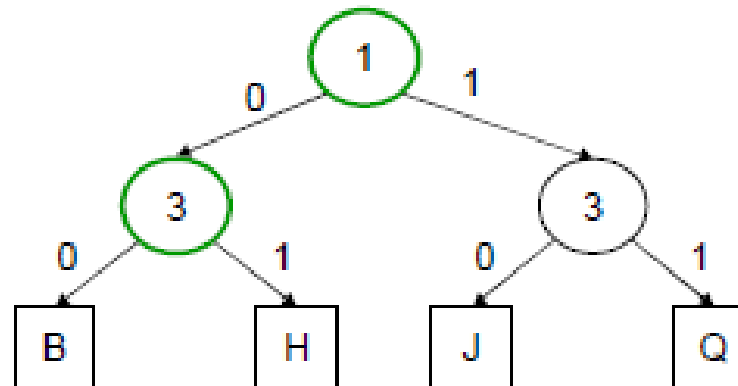
Inserção $X = Q$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

J e Q seguem o padrão
1xxxxx.
O 3o. bit diferencia J de Q

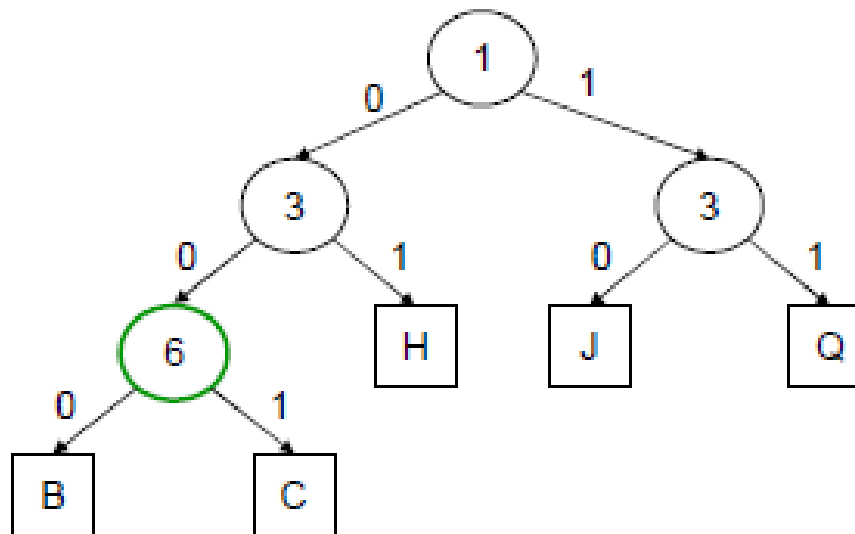
Inserção $X = C$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

B e C seguem o padrão
0x0xxx.

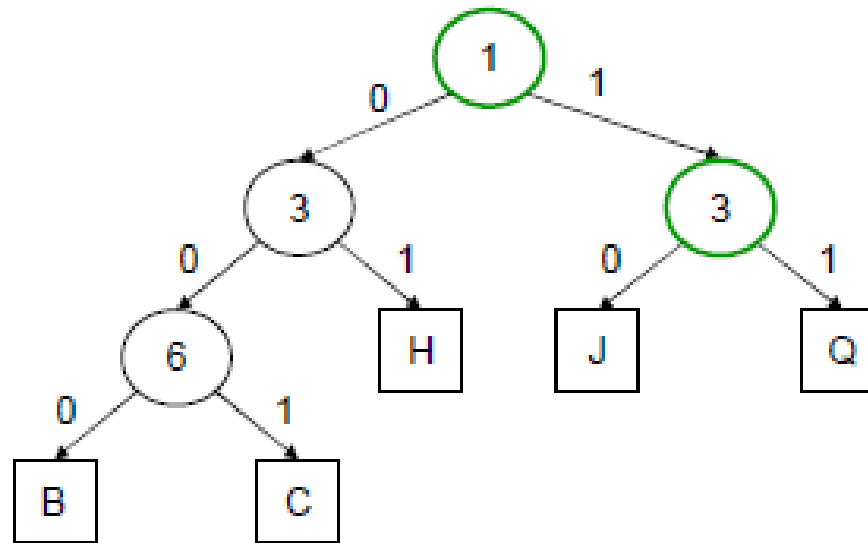
Inserção $X = C$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

B e C seguem o padrão
0x0xxx.
O 6o. bit diferencia B de C

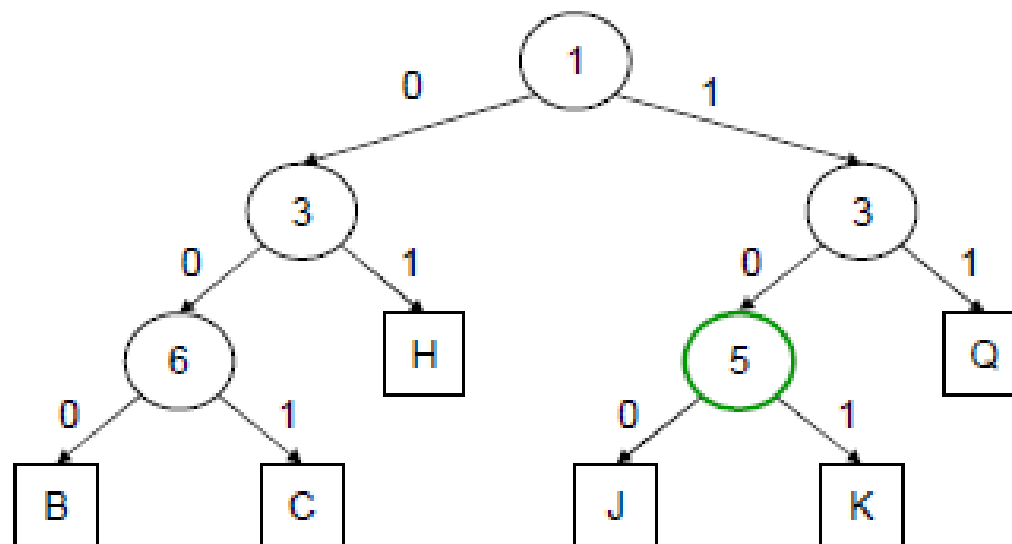
Inserção $X = K$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

J e K seguem o padrão
1x0xxx.

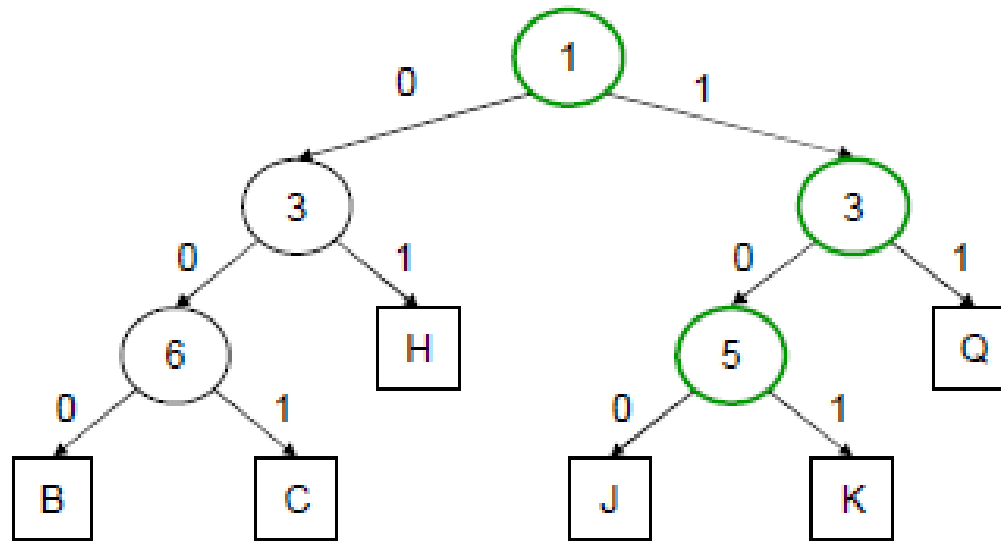
Inserção $X = K$



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

J e K seguem o padrão
1x0xxx.
O 5o. bit diferencia J de K

Inserção $X = W$

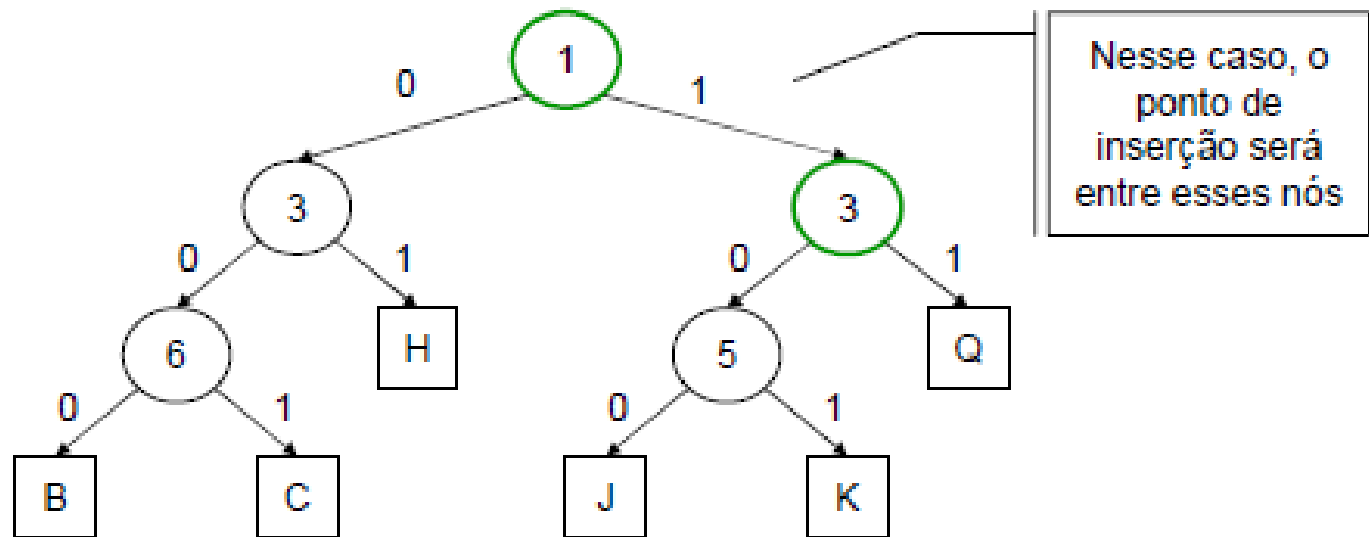


B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

K e W seguem o padrão
1x0x1x.
O 2o. bit diferencia K de W

W=110110

Inserção $X = W$



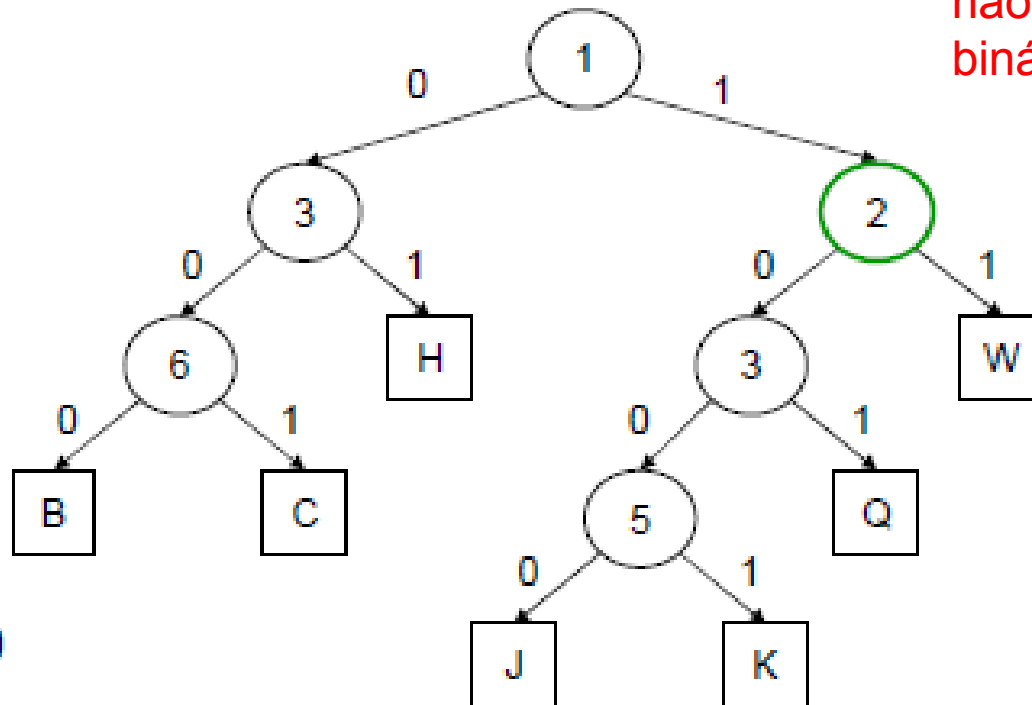
B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

K e W seguem o padrão
1x0x1x.
O 2o. bit diferencia K de W

W=10110

Inserção $X = W$

Obs: A árvore PATRICIA não é necessariamente binária!



B = 010010
C = 010011
H = 011000
J = 100001
Q = 101000
K = 100010

W=110110

Árvore Patrícia: Algoritmo de inserção

1. Se a subárvore 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 subárvore atual for simplesmente um nó de informação, os bits da chave X são comparados, a partir do bit de índice imediatamente após o último índice da seqüência de índices consecutivos do caminho de pesquisa, com os bits correspondentes da chave X deste nó de informação, até encontrar um índice **i cujos bits sejam diferentes**
 - A comparação dos bits 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

Árvore Patrícia: Algoritmo de inserção

3. Se a raiz da subárvore atual for um nó de desvio, deve-se prosseguir para a subárvore indicada pelo bit 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 segundo a chave X . A seguir, o nó de desvio é ligado ao de informação pelo ponteiro de subárvore esquerda ou direita, dependendo se o bit de índice i **da chave X seja 0 ou 1, respectivamente**
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

Aplicações das TRIES

Dicionários (telefone celular)

Corretores Ortográficos

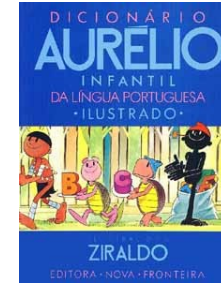
Programas para compreender Linguagem Natural

Auto-preenchimento:

browsers,

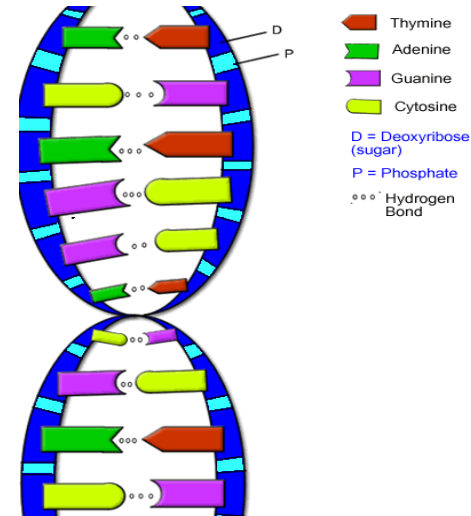
e-mail,

linguagens de programação



Aplicações das TRIES

- * **Compressão** de dados
- * **Biologia** computacional



- * Tabelas de **roteamento** para endereços IP
- * Armazenar e consultar **documentos XML**
- * Fundamental para o Burtsort (o método mais rápido de ordenação de strings em memória/cache)
- * Tabelas de **símbolos** em compiladores

Exercício

Insira as seguintes chaves em uma Árvore Patrícia:

A 000100

B 010100

C 000010

D 100100

E 001001

F 001100

G 101000

H 101010