UNIVERSIDADE FEDERAL
DE ALAGOAS

Trie

Equipe: Jadson César, João Victor, Rafael Galhós e Rodrigo Alves

https://github.com/Projeto-Estrutura-de-Dados

Problema

Imagine a situação em que você está conversando algo importante com alguém e precisa mandar uma grande quantidade de informações para essa pessoa, e esta pessoa não pode ouvir áudios no momento. Não seria ótimo se o teu celular completasse as palavras para você?

E se você fosse fazer uma pesquisa de uma palavra em um dicionário ou em um grande texto?

Se armazenarmos chaves na árvore de pesquisa binária, um ABB bem equilibrado precisará de um tempo proporcional a M * log N, onde M é o comprimento máximo da string e N é o número de chaves na árvore.

Consegue imaginar essa pesquisa sendo feita utilizando a estrutura de árvore, com o tempo de pesquisa de O(M), sendo M o tamanho da chave?

Algoritmo trie

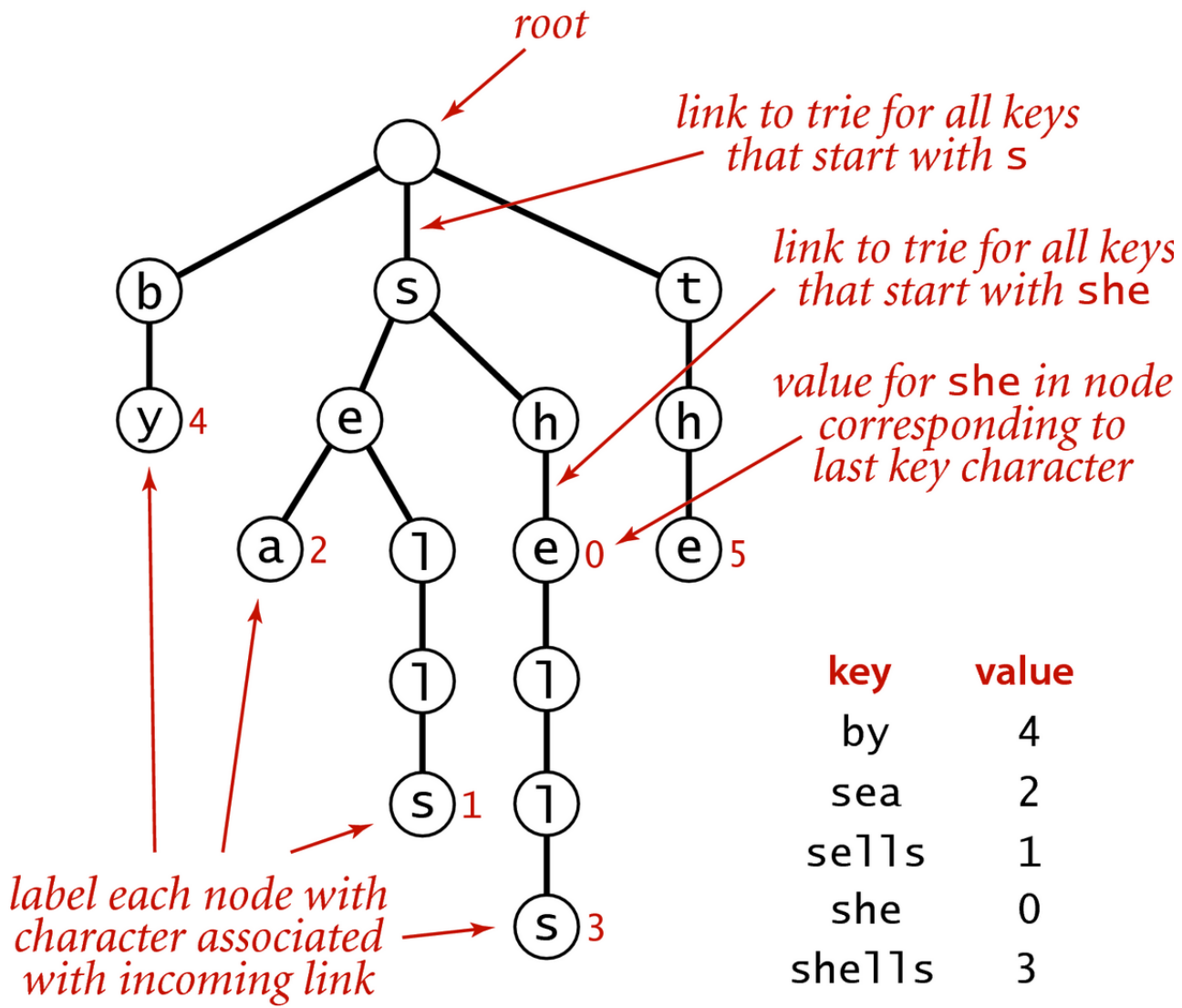
- **O que é?**

Uma trie é uma estrutura em árvore que pode servir para guardar, por exemplo, registos ou valores como num array associativo, em que uma dada chave(palavra) corresponde a um valor, ou então um simples array sequencial.

Uma característica interessante é que a Trie armazena dados de forma não explicita, armazenando a ultima letra e fazendo o caminho de volta até a raiz tem se a palavra completa inserida.

Definições

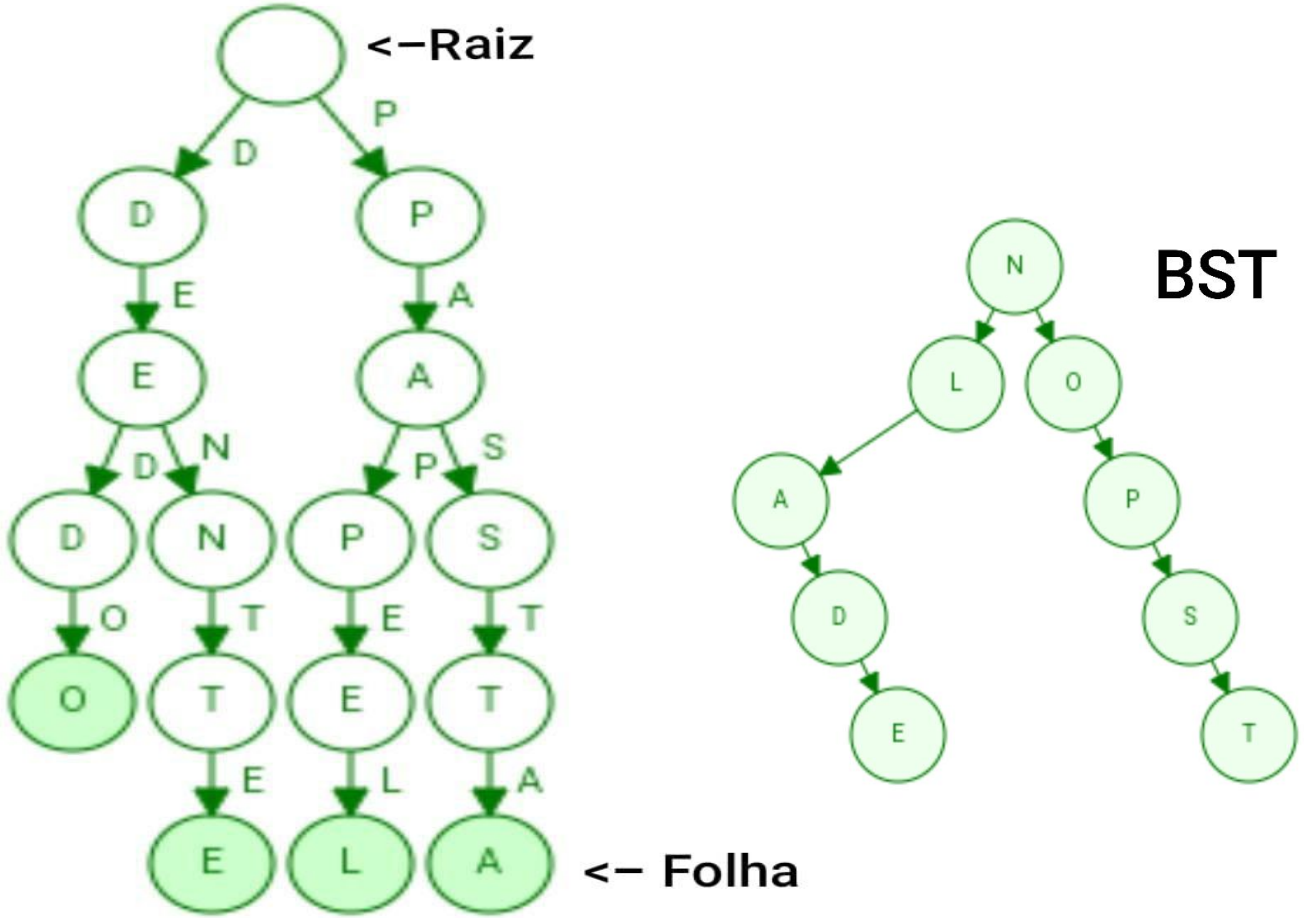
- **Raiz** – É o primeiro nó da árvore é a partir dela que a árvore se ramifica e outra característica é que é o único nó que não é apontado por outro.
- **Subtrie** – É um nó sem ser a raiz onde surge ou pode surgir ramificações, qualquer nó de uma trie é uma subtrie.
- **Folha** – É um nó que representa a última letra de uma palavra.



key	value
by	4
sea	2
sells	1
she	0
shells	3
the	5

Anatomy of a trie

Chaves: Dedo, dente, papel e pasta.



BST

DEFININDO A STRUCT DA TRIE

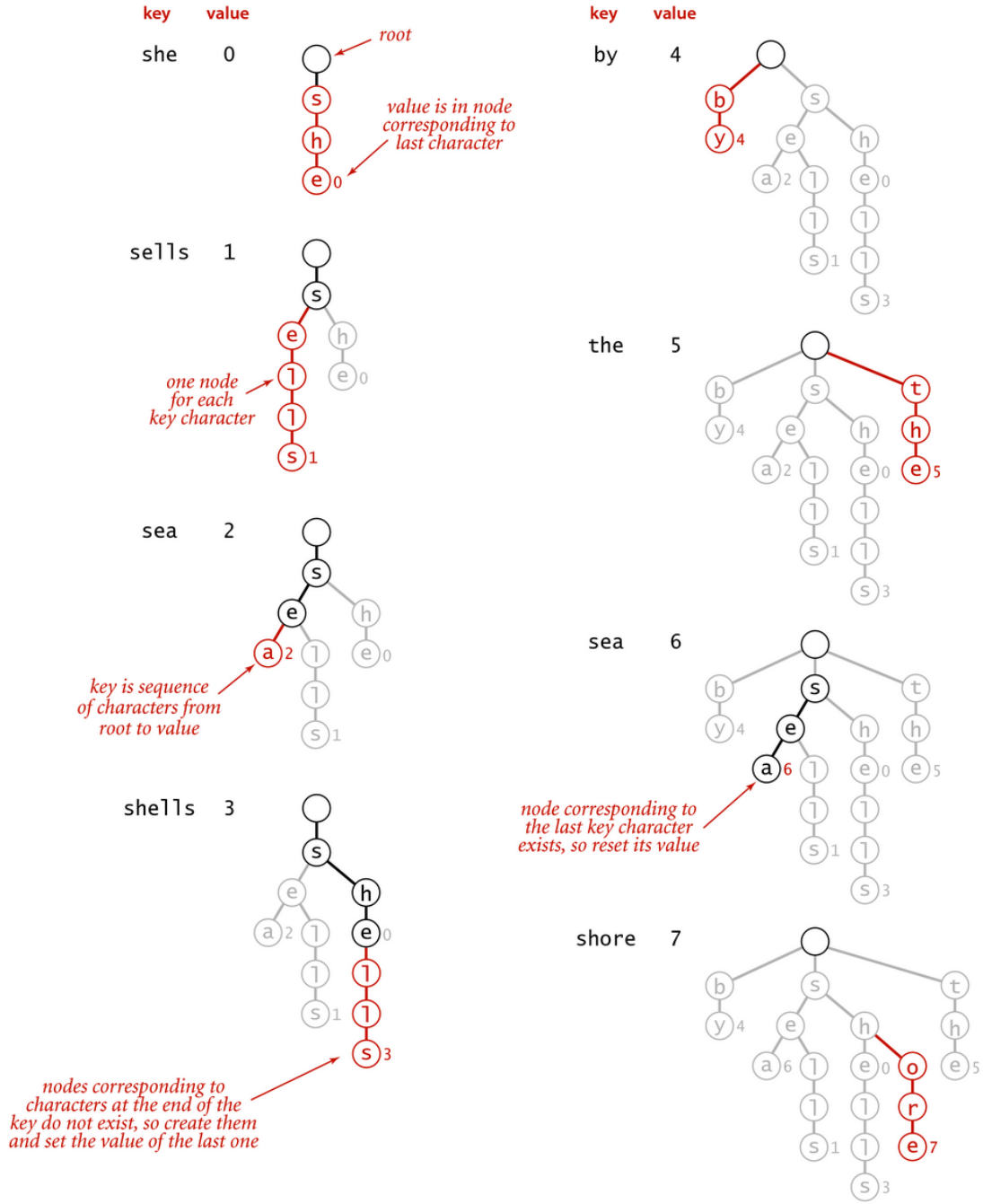
```
#define CHAR_SIZE 26 // Tamanho do alfabeto com letras minúsculas
struct Trie
{
        int isLeaf;     // Quando for = 1, é folha.
        struct Trie* character[CHAR_SIZE];
};
```

inserir

```
void insert(struct Trie *head, char* str)
{
        struct Trie* curr = head;                // inicia a partir do nó raiz

        while (*str != '\0')
        {
                if (curr->character[*str - 'a'] == NULL)  //cria um no se o caminho não existir
                        curr->character[*str - 'a'] = newNode();

                curr = curr->character[*str - 'a'];              // aponta pro próximo nó
                str++;                                   // anda para o próximo caractere
        }
        curr->isLeaf = 1;                                // marca a última letra da palavra
}
```



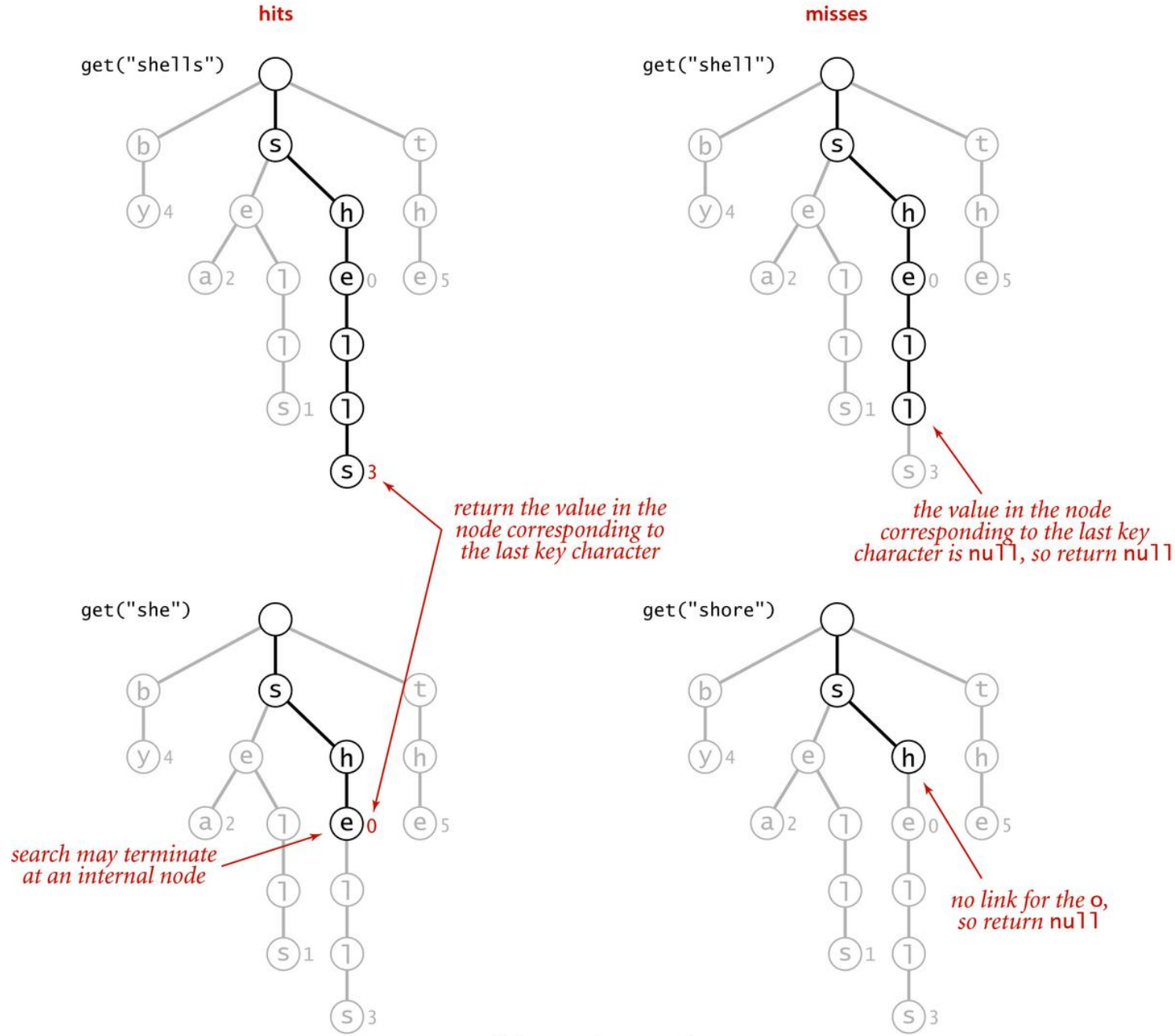
Trie construction trace for standard indexing client

busca

```
int search(struct Trie* head, char* str)    //Retorna 1 se a string for encontrada na Trie, se não
                                                ela retornará 0
{
        if (head == NULL)                   // Retorna 0 se a trie é vazia
            return 0;

        struct Trie* curr = head;

        while (*str != '\0')
        {
                curr = curr->character[*str - 'a'];                 //Vai para o próximo nó
                if (curr == NULL)     // Se a string for inválida (final do caminho alcançado em Trie)
                    return 0;
                str++;                              // Anda para o próximo caractere
        }
        return curr->isLeaf;                // Se o nó atual é uma folha e chegamos ao final da string,
                                                retorne 1
}
```



Trie search examples

Simulação de inserção, busca e remoção.

- https://www.cs.usfca.edu/~galles/visualization/Trie.html

Motivação

- **Auto-preenchimento**

São armazenados em tries e a medida que é digitada uma string de caracteres o algoritmo vai comparando a existência de correspondências na estrutura. A cada caractere digitado são apresentadas as opções de preenchimento, e no momento em que só existir um caminho possível a ser seguido na trie ocorre o preenchimento automático.

Motivação

- No caso do dicionário, vimos que a busca de uma palavra na trie não ocorre como numa árvore binária, não navega por metade da árvore, pois a busca usa a posição para pesquisar e não o tamanho do caractere.