

# **Estrutura de Dados**

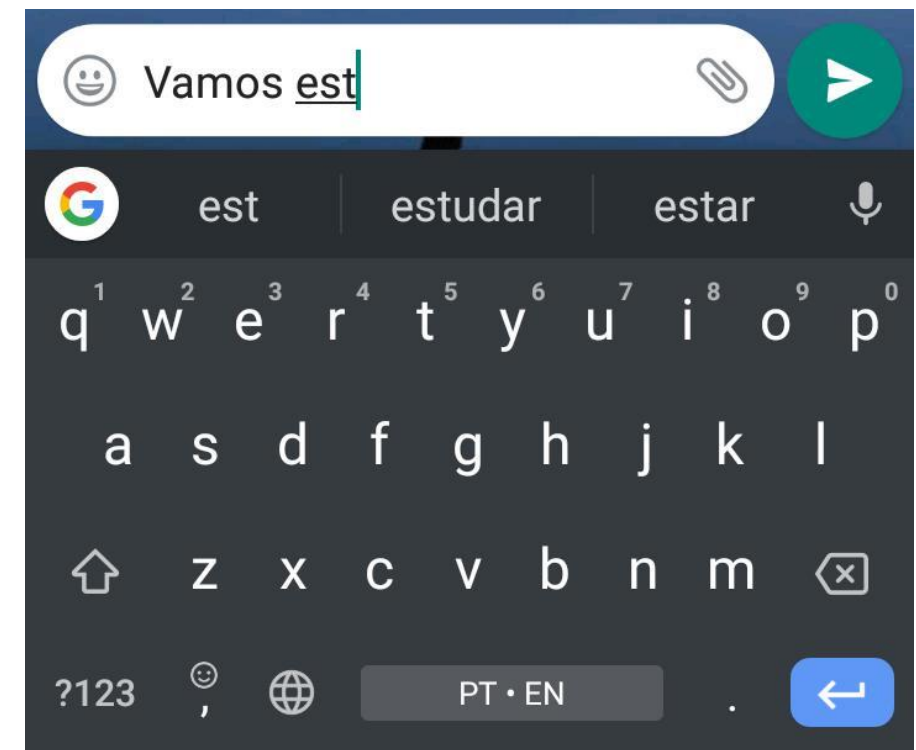
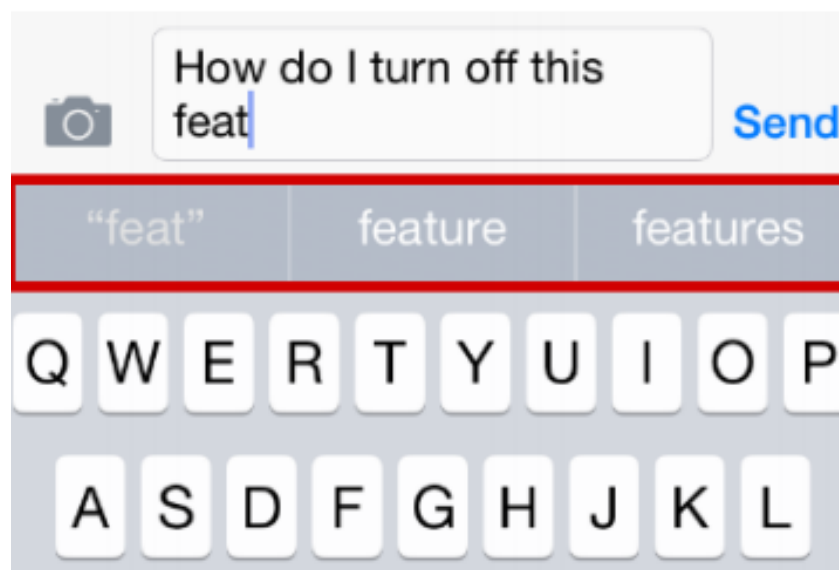
**Equipe : Lucas Alves, Ewerton Felipe e John Davi Dutra**

**<https://github.com/JohnQ00/ProjetoEstruturadeDados>**

Como funciona o autopreenchimento ?  
E a correção automática ?

estrutura de da|

estrutura de dados  
estrutura de dados **arvore**  
estrutura de dados **fila**  
estrutura de dados **em java**  
estrutura de dados **em python**  
estrutura de dados **em c**  
estrutura de dados **lineares**  
estrutura de dados **e algoritmos em c++ adam drozdek**  
**pdf**  
estrutura de dados **pilha**  
estrutura de dados **e algoritmos em c++ pdf**



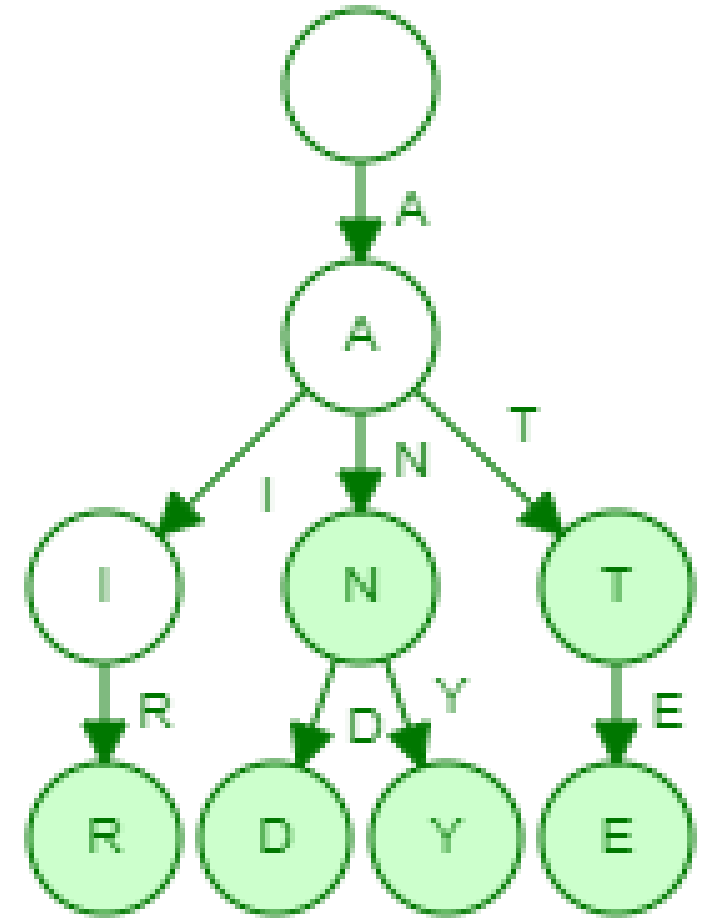
De que forma podemos fazer isso ?

Usando Hash ?

Usando Array ?

# Trie

- A trie é uma estrutura ideal para a retomar dados anteriores, por isso o nome **retrieval** trie.
  - É caracterizada como uma árvore de busca.
- Usar as strings como **keys** para formar um caminho de busca.
- Também são conhecidas como árvores de prefixo e como árvores digitais.



# Por que usar a Trie ?

- Na trie, a inserção e a busca são classificadas como  $O(n)$ , pois depende exclusivamente do tamanho da string inserida ou buscada.
- A classificação/organização de caracteres é de fácil entendimento para qualquer pessoa.
- Não existe possibilidade de colisão.
- Possui características semelhantes à uma árvore qualquer.

# Definições

- **Internal Node:** Um nó que possui um caractere pertencente a uma palavra adicionada na árvore.
- **Root:** Representa o começo de uma árvore, também podendo representar uma árvore vazia.
- **Path:** Representa a junção de nós com a ponte que os conecta.
- **Edge:** Conecta dois nós.
- **Depth:** É a distância do caminho feito do nó para a raíz.
- **Leaf:** Um nó que não possui filhos.

# Tipos Abstratos de Dados (TAD)

```
struct trie
{
    int isWord;
    trie *word[CHAR_SIZE];
};
```

- **trie \*create\_new\_node();**
- **int haveChildren(trie \*head);**
- **void insert(trie \*head, char \*str);**
- **int search(trie \*head, char \*str);**
- **trie \*delete(trie \*head, char \*str, int depth);**



# Search()

```
int search (trie *head, char *str)
{
    trie *walk = head;
    int k;
    for (k = 0; str[k]; k++)
    {
        if (walk -> word[str[k]] == NULL)
            return 0;
        walk = walk -> word[str[k]];
    }
    if (walk -> isWord == 1)
        return 1;
}
```

# Animação

- Animação Trie

# Então, como a Trie é utilizada ?

- Sabendo que Trie organiza as palavras e as classifica por seus prefixos semelhantes, podemos entender como o autopreenchimento em qualquer mídia funciona.

