

Universidade Federal de Itajubá – *Campus* Sede

Algoritmos e Estrutura de Dados III

Professora Vanessa

*Árvore TRIE*

*Trabalho 2*

Jonas Henrique, matrícula 34397

Samantha Aquino, matrícula 35027

Junho/2018

**ÁRVORE TRIE**

**1. INTRODUÇÃO**

As Árvores Trie foram definidas por Edward Fredkin em 1960, e a vêm de Re**trie**val – que está relacionado à recuperação de informações. Conhecido também como árvore de busca digital e como árvores de prefixos.

Uma trie é útil quando o conjunto de chaves é denso, fazendo com que a maioria dos ponteiros em cada nó é usada. Quando o conjunto de chaves é esperado, uma trie desperdiça uma grande quantidade de espaço com grandes nós que estão, em sua grande maioria, vazios. Caso for conhecido o conjunto de chaves antecipadamente, e não ter alterações, existe diversas técnicas que podem ser aplicadas para diminuir as exigências de espaço. Uma das técnicas é de estabelecer uma ordem diferente na qual os símbolos de uma chave são utilizados para pesquisar (por exemplo, de modo que o terceiro símbolo da chave de argumento pudesse ser usado para acessar o ponteiro apropriado na raiz da trie, o primeiro símbolo nos nós do nível 1, e assim por diante. (TANEBAUM, LANGSAM, AUGENSTEIN 1995, p. 593).

Abaixo será estudado melhor sobre a árvore trie, sua implementação e a seguir a proposta do trabalho.

* 1. **Árvores trie**

Uma árvore trie, do inglês reTRIEval (recuperação), é um tipo de árvore N-ária, onde os descendentes do mesmo nó têm o mesmo prefixo. O caminho da raiz para qualquer nó é um prefixo de uma chave. A árvore trie é mais eficiente quanto maior quantidade de chaves com prefixos comuns.

Trie não armazena chaves na árvore, as chaves estão nos nós externos da árvore e para inserir uma chave, percorrer o caminho descrito pelo padrão de bits da chave, armazene a chave no nó externo no final do caminho. Sua complexidade: é de log(N), a altura da árvore é limitada pelo número de bits nas chaves e se tivermos chaves maiores, a altura aumenta.

As árvores 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;
* Expressão regulares (padrões de pesquisa).

Em geral as árvores tries são utilizadas quando se encontra uma grande quantidade de textos estáticos, isto é, textos que não sofrem alterações ou quando novos documentos não são inseridos. Por exemplo, a indexação de dicionário ou livros. (SANTOS, p.11)

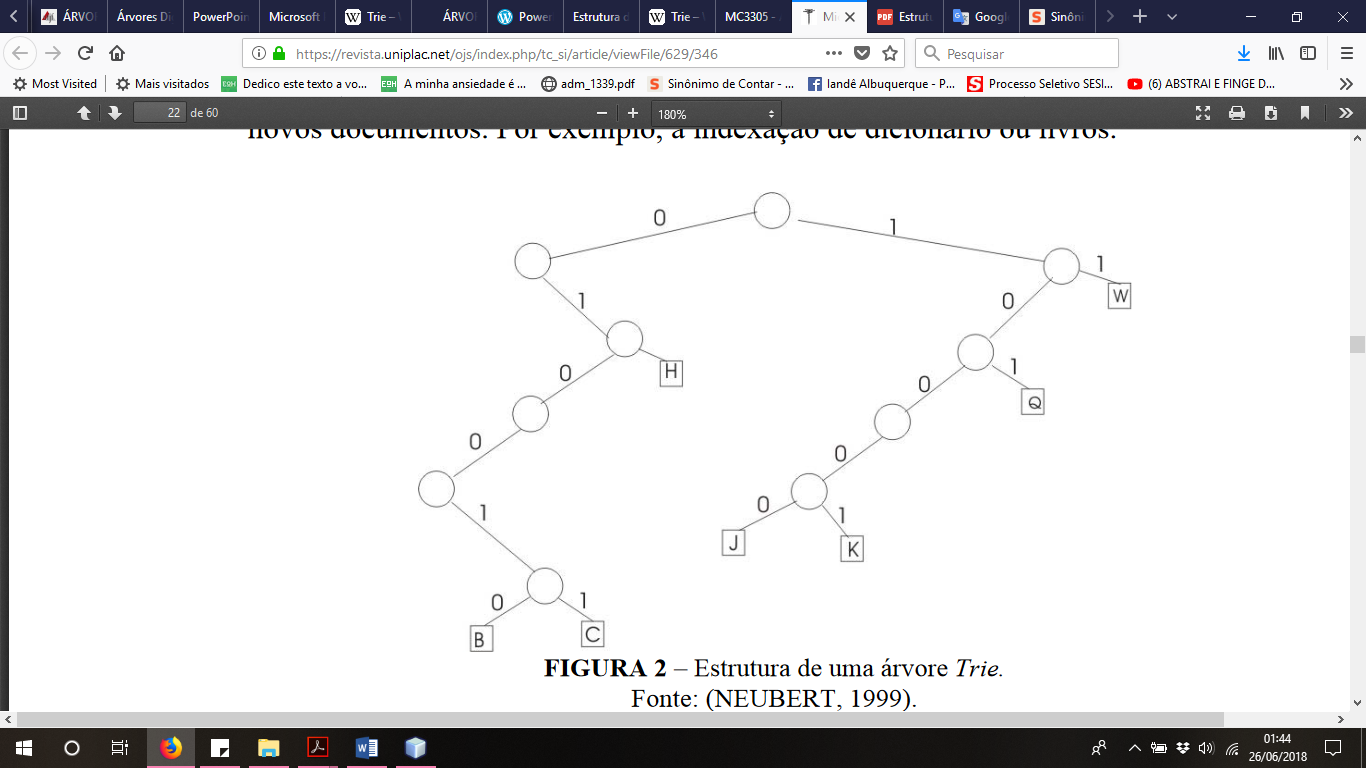
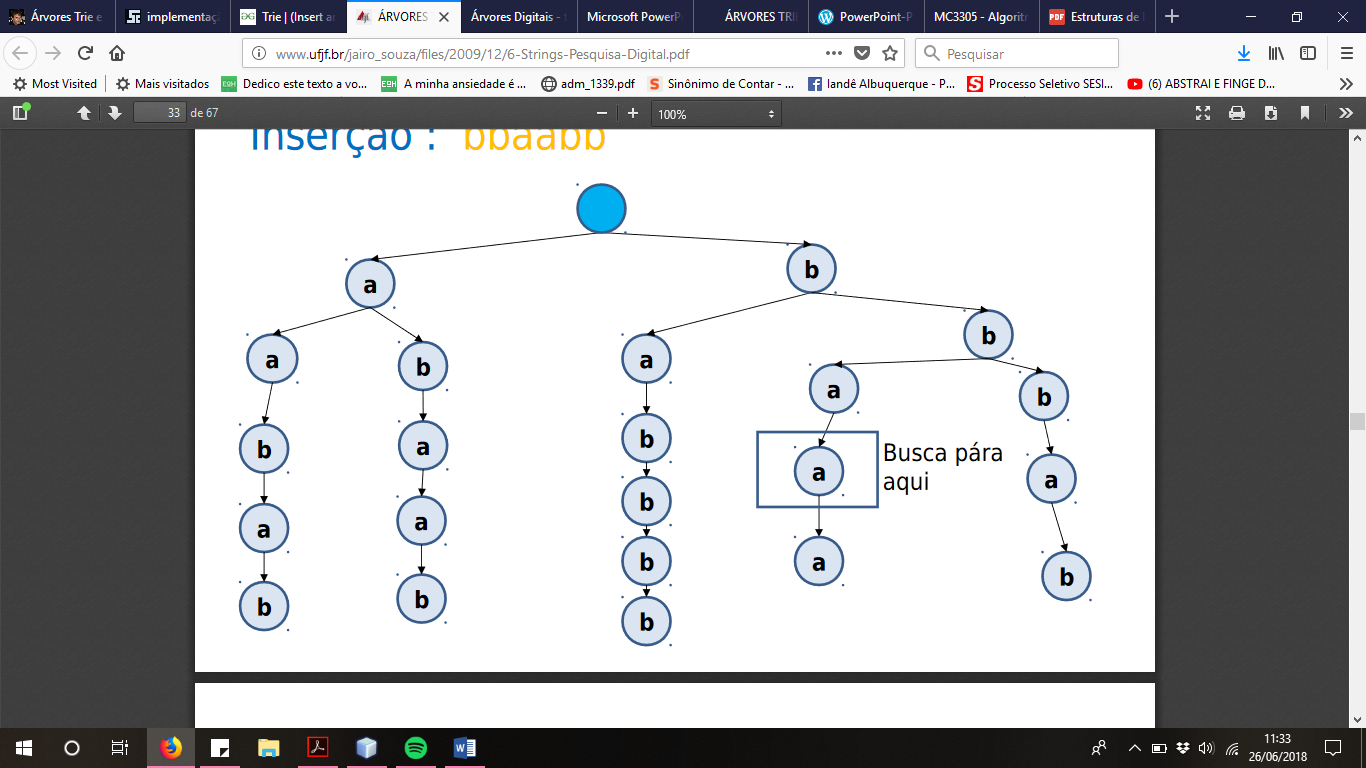


Figura x – Estrutura de uma árvore trie.

1. **DESENVOLVIMENTO**
   1. **Operações**
      1. **Inserção**

Faz-se uma busca pela palavra a ser inserida. Caso ela já exista na árvore trie, nada é feito. Caso contrário, é recuperado o nó até onde acontece a maior substring da palavra a ser inserida e o restante de seus caracteres são adicionados na árvore trie a partir daquele nó. Como exemplo de inserção em uma árvore trie temos a figura x a seguir que mostra onde seria inserido a palavra *bbaabb*.



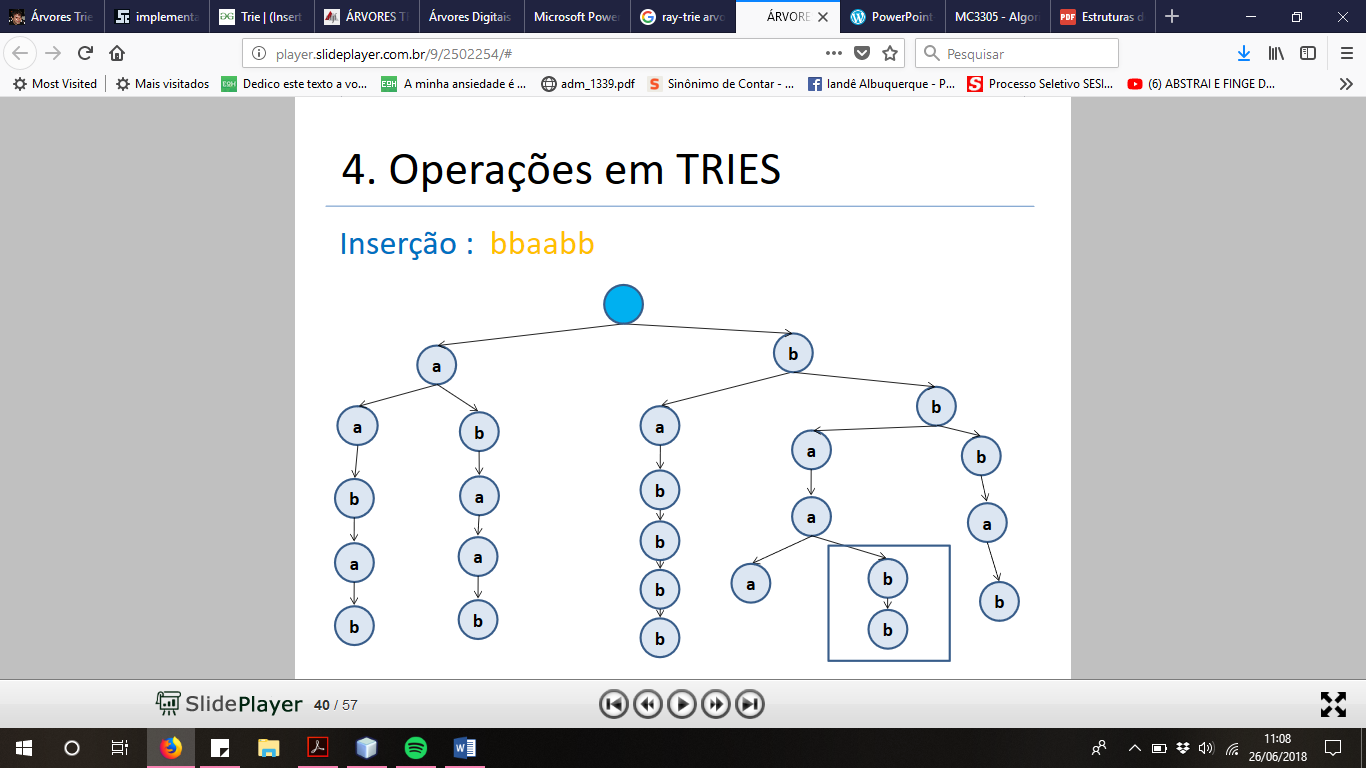


Figura x – Inserção em árvore trie.

* + 1. **Busca**

Para buscar uma certa palavra vai de caractere a caractere para ver se a chave pertence a árvore trie. Seguindo os passos abaixo:

1. Busca no nível superior o nó que confere com o primeiro caractere da chave.
2. Se não acha nenhum, retorna falso.
3. Senão, se o caractere que confere é igual retorna verdadeiro.
4. Move para a sub árvore trie que confere com o caractere.
5. Avança para o próximo caractere na chave.
6. Volta ao passo 1.

Como podemos ver no exemplo da figura x abaixo onde se busca pela palavra *ro.*

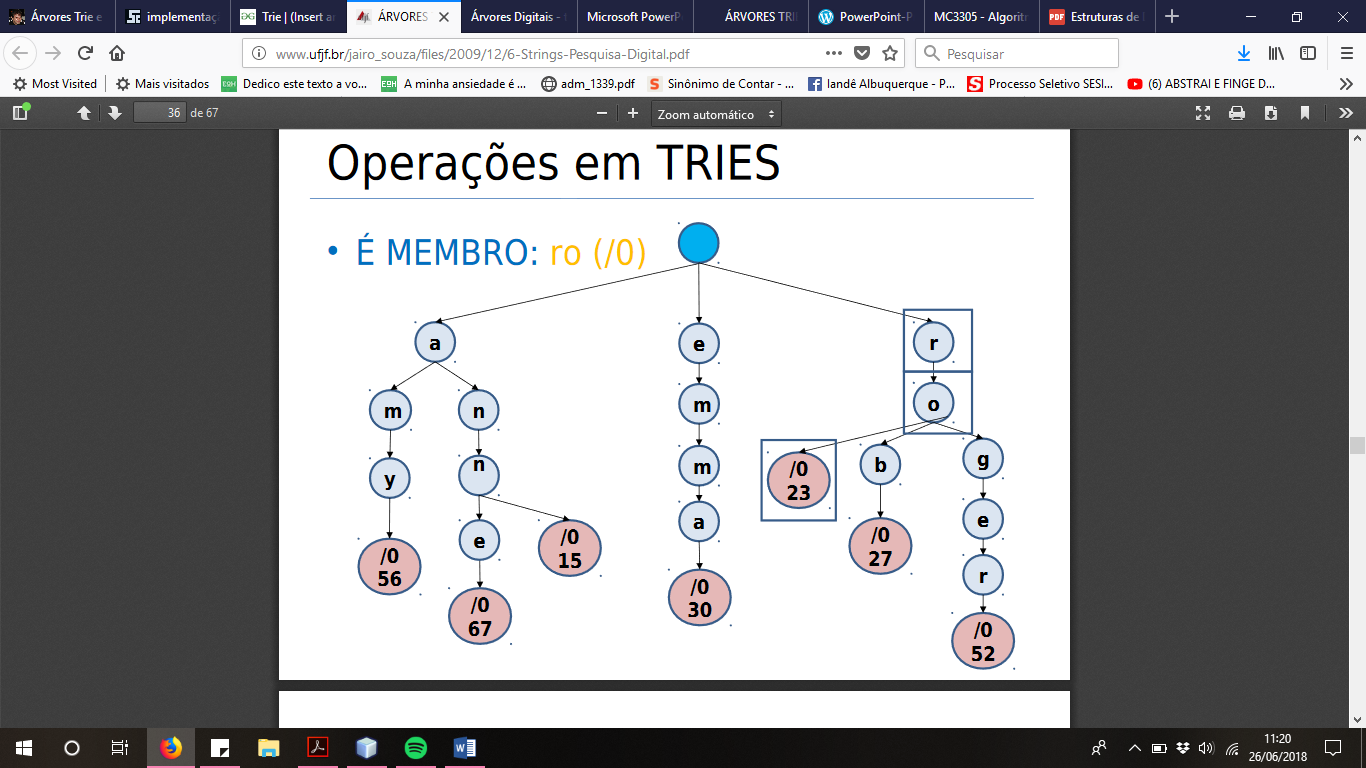
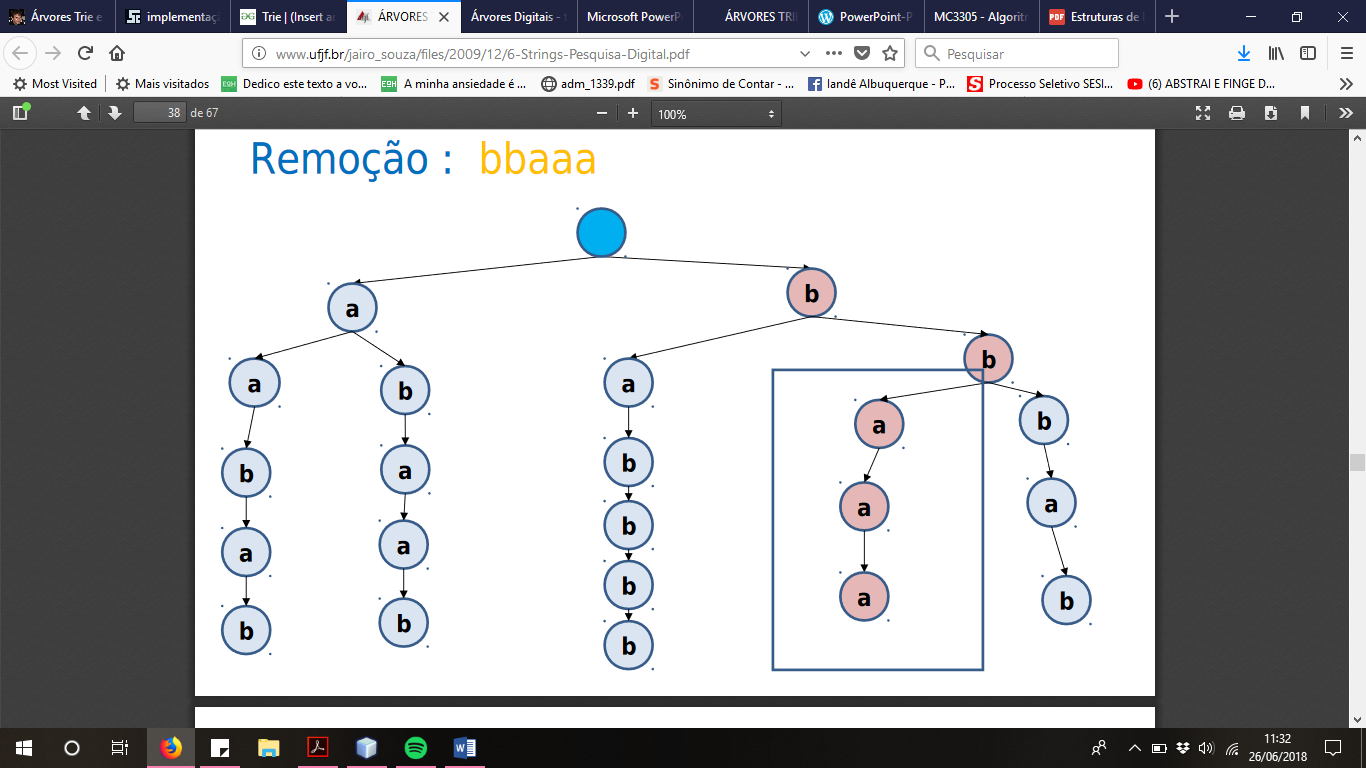


Figura x – Busca em árvore trie.

* + 1. **Remoção**

Para remover uma palavra da árvore trie busca-se o nó que representa o final da palavra a ser removida. E então são removidos os nós que possuem apenas um filho pelo caminho ascendente. É então concluída a remoção apenas quando se encontra um nó com mais de um filho. Como mostra a figura x abaixo onde deseja-se remover a palavra *bbaaa.*



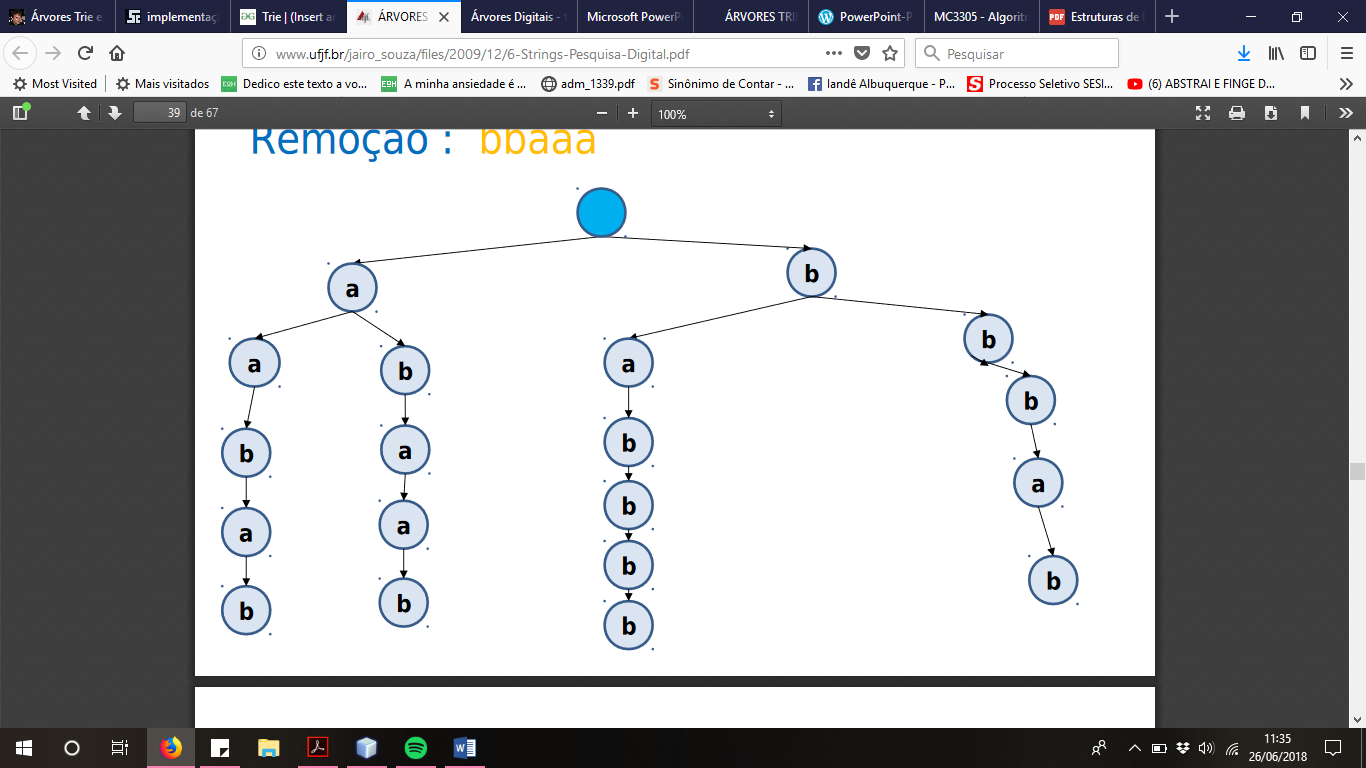


Figura x – Remoção em árvore trie.

* 1. **Implementação**

Para implementarmos o trabalho, foi utilizado a linguagem C. E a estrutura escolhida foi a R-Way, por ser uma implementação mais simples. Na R-Way, a árvore contém dois tipos de nós: o nó de desvio e o nó da informação. Cada nó de desvio contem os valores do tipo alfabeto mais 1 símbolo especial para determinar a chave. Nesse tipo de implementação há um pode de desperdício de espaço.

A árvore r-way trie na figura x contém os dois tipos de nós, o de desvio e o de informação. No nó de desvio contém 27 campos + 1 (b) para determinar uma chave.

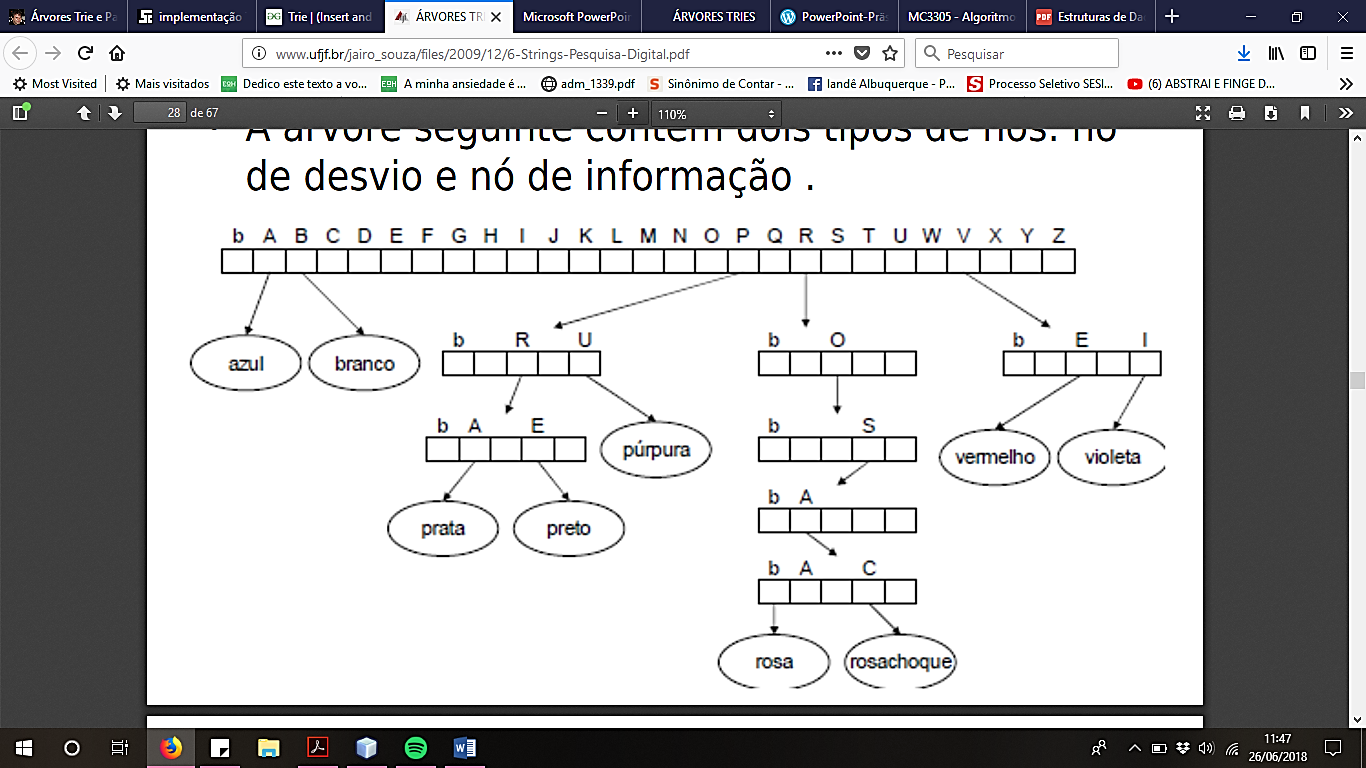
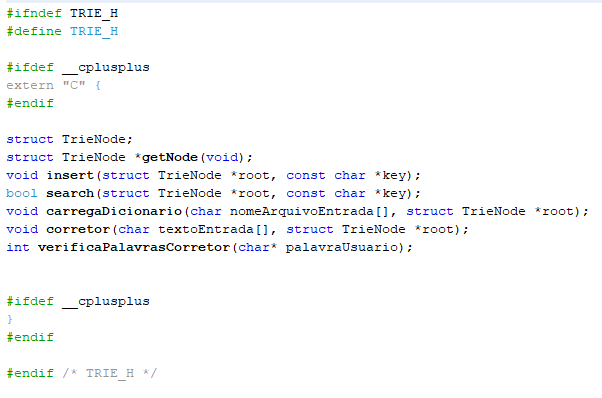


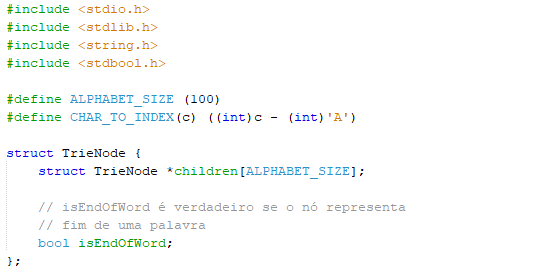
Figura x – R-way

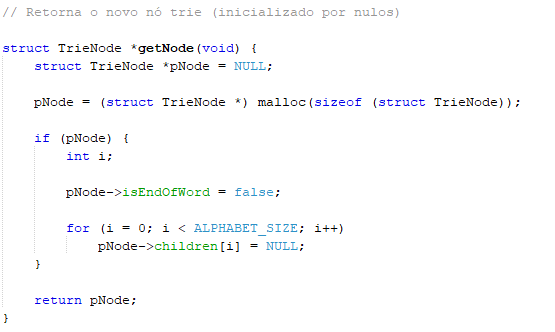
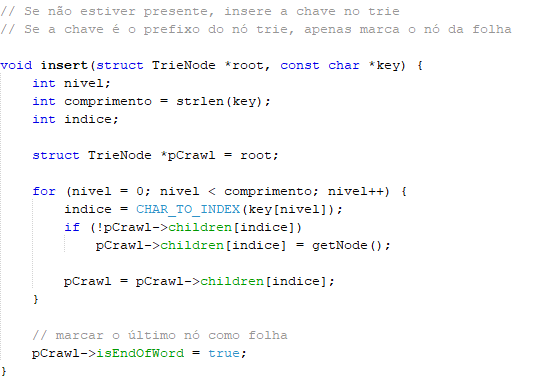
As vantagens é que nas R-Way a busca é randômica em cada em cada nível/caractere, aumentando a velocidade do acesso e não precisa aplicar funções para converter chaves alfabéticas em numéricas. E como desvantagens nas R-Way os nós próximos as folhas terão subTries como NIL, desperdiçando espaço.

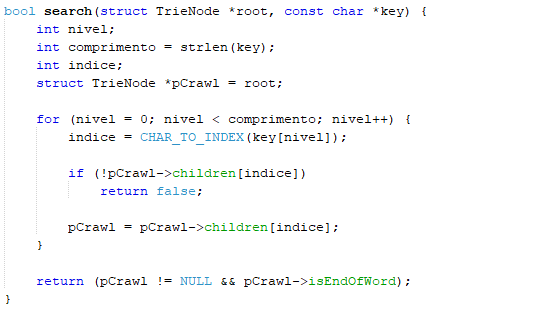
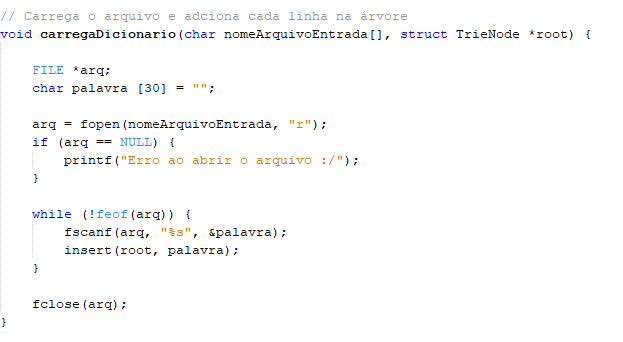
* 1. **Aplicação implementada**

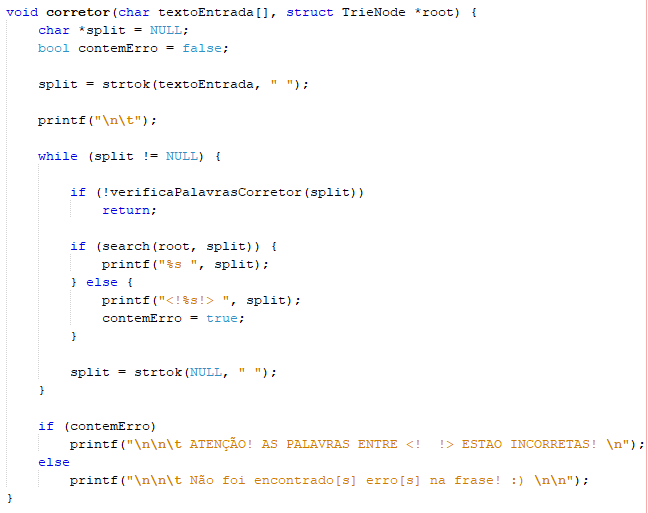
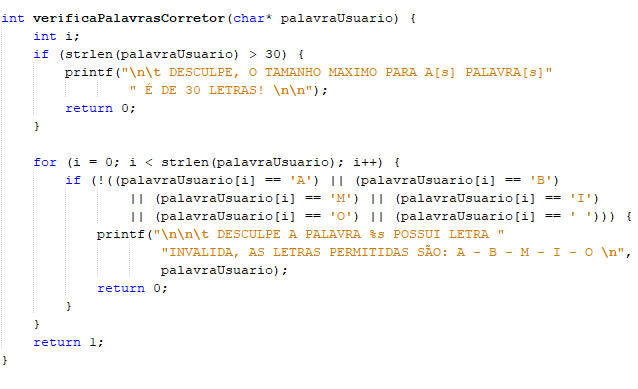
Na implementação, foi criado um arquivo de cabeçalho e um binário com as funções da TRIE e um arquivo binário (main) para interação com o usuário.

Arquivo de cabeçalho: trie.h

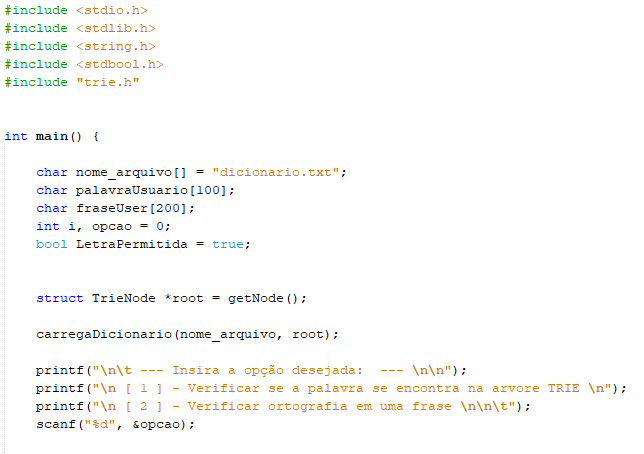
Arquivo binário em C: trie.c

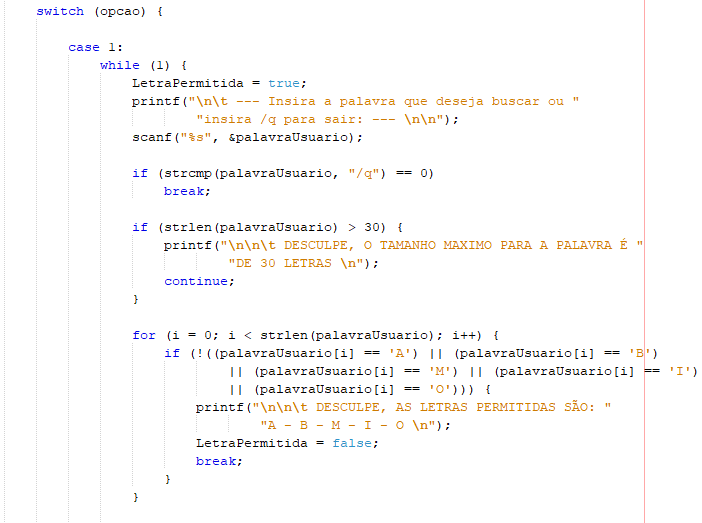


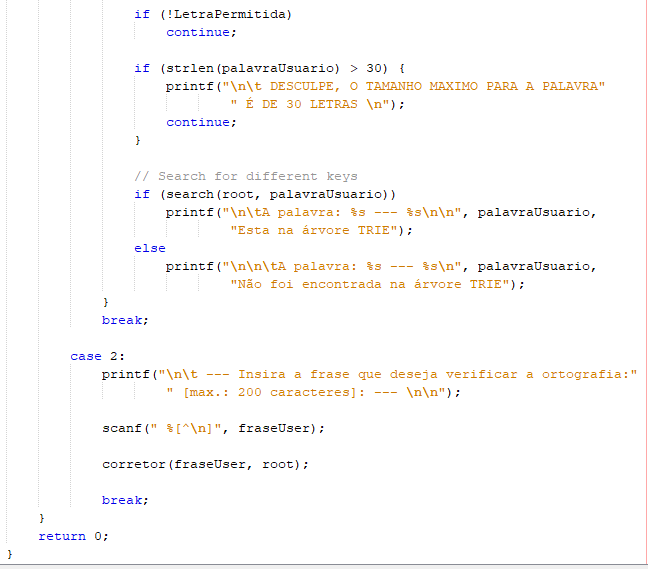




Arquivo binário em C: main.c







# **PRINCIPAIS DETALHES DA IMPLEMENTAÇÃO**

No carregamento da implementação, o programa procura um arquivo dicionário.txt e insere cada linha deste arquivo na árvore TRIE.

Em nossa implementação, o usuário ao iniciar o programa escolhe se quer verificar se a palavra existe na árvore TRIE ou se deseja verificar a ortografia em uma frase. Caso o usuário selecione verificar se a palavra existe na árvore, ele deverá inserir uma palavra com no máximo 30 letras e que contenha apenas as letras A – B – M – I – O, e se ele inserir algo diferente, o programa alertará o usuário e não executa a busca.

Se a palavra estiver de acordo com as regras, irá ser feita uma busca verificando se cada letra existe na árvore e a função retorna TRUE se encontrar ou false caso não encontre. Para a análise, é verificado se cada letra da palavra inserida pelo usuário se encontra na árvore TRIE, se chega no ultima letra da palavra inserida e na arvore esta letra é falsa para fim da palavra, retorna falso, ou caso seja o último nível da arvore e ainda exista letras na palavra, retorna falso.

Caso o usuário escolha a opção de verificar a ortografia em uma frase, realizamos um split na frase inserida utilizando os espaços e verificamos, cada palavra resultante do split, se elas estão de acordo com o padrão de letras e tamanho exigido. Logo após é verificado se cada palavra do split se encontra na arvore TRIE, no caso, nosso dicionário. Se não encontrar a palavra na arvore a palavra é impressa entre “<!” e “!>” para o usuário e caso encontre a palavra é impressa normalmente. Após a análise das palavras, caso seja encontra erro, é informado ao usuário que as palavras entre “<!” e “!>” estão incorretas, se não, é informado que a frase esta correta.

1. **CONCLUSÃO**

Conclui-se que com esta arvore, conseguimos usa-la em varias aplicações de texto que procuram utilizar *autocomplite* e/ou mesmo verificar se a palavra está correta, e o tempo de busca da palavra é rápida, a variação utilizada R-Way é de fácil compreensão e entendimento apesar de possuir desperdício de espaço.

**REFERÊNCIAS**

CAVA, U. LUZZARDI, P. **Árvores tries.** Disponível em: [http://player.slideplayer.com.br/9/2502254/#](http://player.slideplayer.com.br/9/2502254/) . Acesso em 25 de junho de 2018.

SOUZA, J. **Busca digital (TRIE e Árvore patrícia).** Disponível em: <http://www.ufjf.br/jairo_souza/files/2009/12/6-Strings-Pesquisa-Digital.pdf>. Acesso em 25 de junho de 2018.

SANTOS, I. **Indexação de arquivos texto: Uma contribuição para o projeto openged.** UNIPLAC. 2012. Disponível em: https://revista.uniplac.net/ojs/index.php/tc\_si/article/viewFile/629/346. Acesso em 25 de junho de 2018.

SZWARCFITER, J. L., MARKENZON, L. **Estruturas de Dados e seus Algoritmos**. 3a ed. Rio de Janeiro: LTC, 2010.

TENENBAUM, A. M., LANGSAM, Y., AUGENSTEIN, M. J. **Estruturas de Dados Usando C.** São Paulo: Makron, 1995.