# Decifrando os Segredos de Arendelle

**Guilherme de Abreu Lima Buitrago Miranda - 2018054788**

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

guilhermemiranda@dcc.ufmg.br

## 1. Introdução

O trabalho prático 3 da disciplina de Estrutura de Dados baseia-se na decodificação de mensagens em código morse para o Serviço Secreto Arendellense (SSA) de maneira computacional, uma vez que antigos documentos foram encriptados dessa maneira e atualmente poucos profissionais ativos são capazes de fazer esse trabalho.

Para isso, foi desenvolvido um programa para, dado um arquivo com os caracteres e sua respectiva codificação (ex: $A$ .. $-$ .), recebe um arquivo encriptado e gera uma saída decodificada. A solução computacional desenvolvida foi uma modificação de uma árvore digital ($Trie$) em que todos os caracteres são inseridos baseado em sua codificação.

Posteriormente, quando a mensagem é recebida, os respectivos códigos são buscados na árvore nível a nível, em que cada nível $i$ representa o sinal . ou $-$ presente naquela posição da chave buscada. Ou seja, na chave .. $-$ ., quando $i$ = 1, é feita a busca pelo sinal . naquele nível da árvore, caminhando para a esquerda ou para a direita conforme as regras estabelecidas e explicadas posteriormente.

## 2. Implementação

Para a decodificação das mensagens conforme especificado no enunciado, primeiramente mostrou-se necessário a implementação das classes $BTree$ e $Node$, uma vez que foi empregado o paradigma de orientação a objetos e parte da solução utiliza a abstração de uma árvore digital ($Trie$) modificada.

Além disso, foi também criado uma $struct\ Cell$, que contém uma chave e uma letra, representando, portanto, um caractere e seu respectivo código morse. Dessa forma, uma árvore contém vários nós e cada nó, por sua vez, contém uma célula.

Por conseguinte, mostrou-se necessária a criação de métodos para inserção e busca na $Trie$, implementados como $insert\_cell$ e $find\_cell$. No método de inserção, a seguinte lógica é utilizada: primeiramente, se a célula a ser inserida tem chave do tamanho do nível atual da árvore, o mesmo é designado para o nó atual. Caso contrário, se o caractere daquele índice da chave for um ponto, o método é chamado de maneira recursiva para a esquerda, e se for um traço, para a direita, aumentando o valor do índice nas chamadas e, portanto, do nível da árvore. Não obstante, se o método entende que é necessário caminhar para a direita ou para a esquerda mas o caminho ainda está vazio ($nullptr$), é criado um nó vazio, que poderá ser designado futuramente para outra letra se necessário.

Já no método de busca a lógica desenvolvida foi ainda mais simples. Além de procurar à esquerda caso ponto e à direita caso traço com chamadas recursivas aumentando o índice, o método verifica se o tamanho da chave é igual ao índice, indicando que

chegou-se ao nível correto da $trie$ e, caso seja verdadeiro, retorna a letra da célula do nó atual.

Por fim, além de construtor e destrutor, as classes ainda tem um método que imprime a árvore de maneira pré ordem. Nele, primeiramente verifica-se se o nó atual é nulo e, caso contrário, é feita a impressão do conteúdo da sua célula, letra e chave respectivamente. Posteriormente, duas chamadas recursivas são feitas: a primeira para a esquerda e a segunda para a direita.

Já no $main$, dois procedimentos principais foram desenvolvidos: $receives\_file\_input$ e $decode\_message$. O primeiro utiliza da classe $ifstream$ do $C$++ para ler o arquivo $morse.txt$, que deve estar na pasta raiz junto aos outros arquivos do programa. Com isso, até que atinja-se o final do arquivo, são feitas chamadas de inserção na árvore, passando como argumento uma célula que contém a letra e o código lido naquela iteração.

No segundo procedimento, a leitura da mensagem se dá por meio da entrada padrão. Até que não exista mais linhas a serem lidas, as mesmas são divididas em várias $substrings$, utilizando o espaço em branco para dividi-las, já que é o caractere que separa as letras dos arquivos codificados. Os comandos utilizados para realizar a divisão na $string$ foram retirados de fontes da internet [1]. Assim, é feita a busca na árvore para cada código morse lido, retornando a letra correspondente. Ademais, se o caractere / é encontrado, um caractere em branco é gerado na saída, já que o mesmo indica a separação entre palavras.

Na função $main$ em si, além da criação da árvore e das chamadas para os dois procedimentos supracitados, existe uma verificação para observar se a $flag\ -a$ foi passada como parâmetro na execução do programa. Se verdadeiro, é chamado o método que imprime a árvore usando o caminhamento em pré-ordem.

Para a entrada, o formato esperado é de uma mensagem em código morse em que o espaço em branco representa a separação de letras e a barra / representa a separação entre palavras. Para a saída, é esperada a mensagem decodificada. Por fim, o arquivo a ser lido para a criação da árvore ($morse.txt$) deve ter o caractere, um espaço em branco e, posteriormente, sua chave em código morse, seguidos por uma quebra de linha para a próxima letra a ser inserida na árvore.

O compilador utilizado para o desenvolvimento do programa foi o $g$++/$gcc$ $8.3.0$, usando a flag $-std = c$++$11$ numa máquina com a versão 4.19.49 do Kernel Linux.

## 3. Instruções de compilação e execução

Para a compilação e execução do programa, é necessário, primeiramente, entrar no diretório $/src$ do trabalho. Posteriormente, um comando $make$ é capaz de compilar todos os arquivos necessários para a execução do programa. Por fim, a execução é chamada segundo o seguinte comando: $./tp3 < caso1.in > caso1.out$ ou $./tp3 - a < caso1.in > caso1.out$, sendo a $flag\ -a$ opcional para a impressão da árvore seguindo o caminhamento pré-ordem. A saída esperada é a mensagem decodificada utilizando a árvore construída pelo programa.

[1] url//https://stackoverflow.com/questions/14265581/parse-split-a-string-in-c-using-string-delimiter-standard-c

## 4. Análise de Complexidade

Dados que o arquivo de texto $morse.txt$ é sempre o mesmo, temos que a criação da árvore também acontecerá sempre da mesma forma. Assim, pode-se considerar que o custo para a construção da Trie é sempre constante.

Dessa forma, temos que a única possibilidade de influência no tempo da execução do programa é o tamanho da entrada, que deve ser procurada letra por letra na árvore. Por conseguinte, o único fator a ser observado é a quantidade $n$ de caracteres codificados na entrada e, assim, tem-se que a complexidade do algoritmo é linear, ou seja, $O(n)$.

## 5. Conclusão

O desenvolvimento do algoritmo proposto foi bastante interessante para revisar a criação da Estrutura de Dados "Árvore", além da pesquisa de dados numa Trie. Não obstante, foi útil para praticar mais os conhecimentos na linguagem $C$++ e o paradigma de orientação a objetos.

A maior dificuldade encontrada se deu na modificação da Trie tradicional, sendo necessária a criação de alguns nós vazios, o que representou uma mudança na maneira com que a mesma foi codificada e, portanto, empregado um maior esforço para a abstração.

Isso posto, o trabalho mostrou-se muito interessante para praticar a lógica de programação e aumentar a experiência com a Estrutura de Dados supracitada, além do algoritmo de pesquisa de dados.

## 6. Referências Bibliográficas

Cormen , T., Leiserson, C, Rivest R., Stein, C. Introduction to Algorithms, Third Edition, MIT Press, 2009. Versão Traduzida:Algoritmos – Teoria e Prática 3a. Edição, Elsevier, 2012.

MS LATHA SHILVANTH. Java: For Programming (2018). Createspace independent Publishing Platform. ISBN-10: 1985254603