Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Estruturas de Dados 2019/1
Prof. Luiz Chaimowicz e Raquel Prates
Monitores: Matheus Nunes e Ronaldo Vieira

Trabalho Prático 3: Decifrando os Segredos de Arendelle

Valor: 10 pontos
Data de entrega: 4 de Julho de 2019

Introdução

Desde a última década, o reino de Arendelle vem passando por uma gigantesca revolução tecnológica. Embora ainda predominantemente medieval, soluções tecnológicas são encontradas com facilidade em todos os âmbitos da sociedade arendellense. Lâmpadas de LED com sensores de luminosidade substituem as tradicionais fogueiras noturnas, enquanto no ar, uma vez dominado por pombos-correio, transitam invisíveis ondas de rádio levando acesso à internet aos vilarejos mais remotos.

Essa acelerada ascensão tecnológica, no entanto, tem seu custo. Profissões que antes eram essenciais para o bem-estar público rapidamente se tornaram obsoletas, acarretando em um aumento na demanda por cursos de recapacitação e, sobretudo, na taxa de desemprego do reino (mas isso é assunto para outro trabalho prático).

O Serviço Secreto Arendellense (SSA), órgão encarregado de investigar ameaças, defender a soberania do reino e manter os segredos da família real, costumava usar uma variação do código Morse para fins de comunicação. Atualmente, técnicas mais confiáveis que fazem uso de criptografia são usadas, mas ainda existem nos arquivos do órgão milhares de bancos de dados com mensagens em código Morse. Como a maioria dos profissionais peritos em código Morse foram desligados em regime de aposentadoria voluntária, você foi contratado(a) novamente com o objetivo de traduzir essas mensagens.

Detalhes do problema

O objetivo deste trabalho é praticar os conceitos de árvore e pesquisa binária. Você deverá desenvolver um programa que decodifique textos em código Morse, letra por letra. Para isso, você deverá utilizar uma árvore binária, de forma que cada letra em código Morse seja convertida em uma letra do alfabeto com uma pesquisa na árvore.

0	`-----`	A	`..-.`	K	`-.-`	U	`..`
1	`.----`	B	`-...`	L	`.-..`	V	`...-`
2	`..---`	C	`-.-.`	M	`-.`	W	`.--`
3	`...--`	D	`-`	N	`--.`	X	`-..-`
4	`....-`	E	`.`	O	`---`	Y	`-.--`
5	`.....`	F	`.-`	P	`.--.`	Z	`--..`
6	`-....`	G	`--`	Q	`--.-`		
7	`--...`	H	`....`	R	`.-.`		
8	`---..`	I	`..-`	S	`...`		
9	`----.`	J	`.---`	T	`-..`		

Tabela 1: Variação do código Morse usado pelo SSA.

Cada letra do alfabeto é codificada através de pontos (.) e traços (-) em código Morse. A letra A, por exemplo, é codificada por `..-.` na variação do código utilizada em Arendelle. Além disso, há um espaço entre cada letra e uma barra (/) entre cada palavra. Dessa forma, a frase `LIVRE ESTOU` seria codificada

como `.-..  ..- ...- .-.  . / .  ...  -..  --- ...` A Tabela 1 apresenta a codificação Morse para cada número e letra do alfabeto.

A árvore construída no trabalho deve se assemelhar à definição conhecida de árvore de pesquisa digital. A sua árvore deve ficar organizada de tal forma que, à medida que os pontos e traços do código Morse são lidos, você "desce" na árvore até encontrar o símbolo do alfabeto correspondente. A parte inicial da sua árvore seria algo como o exibido na Figura 1. Neste exemplo, ao receber a sequência de caracteres `.` e `-` seguidos de um espaço, sabe-se que a letra decodificada é `F`.

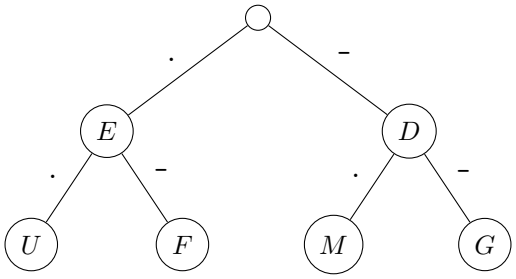

Figura 1: Esquema da árvore a ser construída no trabalho prático.

Seu trabalho consiste, então, em duas partes:

1. **Construção da árvore**: Seu programa deverá ler um arquivo contendo a definição do Código Morse (`morse.txt`) e construir a árvore que será utilizada na pesquisa (decodificação dos caracteres em Morse). Declare e implemente o seu tipo abstrato de dados adequadamente, fazendo funções para inicializar a árvore, inserir nodos, etc. Faça também um procedimento que imprima a árvore (o símbolo e a codificação correspondente) usando o caminhamento em pré-ordem.

2. **Conversão das mensagens**: Com a árvore construída, seu programa deverá ser capaz de converter mensagens em código Morse. Ele deverá ler mensagens em Morse, decodificar cada caractere das mensagens e imprimir na tela as mensagens decodificadas.

Entrada e saída

O formato de entrada e saída desejados são:

Entrada. Neste trabalho, seu programa deverá efetuar três tipos de leituras de entrada diferentes: por arquivo, pela entrada padrão e de parâmetros.

- Leitura de arquivo. Seu programa deverá ler o arquivo `morse.txt` (disponibilizado no Moodle, junto deste enunciado), que contém os números e as letras do alfabeto e suas respectivas codificações em código Morse. A árvore de decodificação deverá ser construída a partir destes dados.

- Leitura da entrada padrão. Seu programa deverá ler mensagens em código Morse através da entrada padrão (`stdin`). As N linhas da entrada conterão as N mensagens em Morse. Cada entrada poderá conter uma quantidade de linhas diferente.

- Leitura de parâmetros. Seu programa deverá aceitar um parâmetro opcional: *Exibir árvore* (-a). Se o parâmetro *Exibir árvore* estiver presente, a saída deverá também conter o caminhamento em pré-ordem da árvore de pesquisa.

Saída. As mensagem decodificadas deverão ser impressas na saída padrão (`stdout`), na mesma ordem em que foram obtidas na entrada, uma em cada linha. As mensagens devem ser compostas apenas por números e letras maiúsculas.

A seguir, caso o parâmetro *Exibir árvore* esteja presente na entrada, seu programa deverá também imprimir o caminhamento em pré-ordem da árvore de pesquisa. Para cada caractere do alfabeto na

árvore, deve ser impresso o caractere e seu código Morse, separados por um espaço. Os conjuntos de caractere e código devem ser impressos um em cada linha.

Exemplos. A Tabela 2 apresenta o modelo e um exemplo de entrada e saída. Neste exemplo, o parâmetro *Exibir árvore* foi utilizado, mas o caminhamento impresso é o resultante da árvore representada pela Figura 1, que está incompleta. Seu programa deverá imprimir o caminhamento resultante da árvore de pesquisa completa.

Após compilado, seu programa poderá ser executado em um terminal de um computador Linux com o comando

`./nomedoprograma`,

inserindo manualmente a entrada e visualizando a saída no próprio terminal, ou

`./nomedoprograma < caso1.in > caso1.out`,

efetuando a leitura da entrada contida no arquivo `caso1.in` e escrevendo a saída no arquivo `caso1.out`, ou ainda

`./nomedoprograma -a < caso1.in > caso1.out`,

também efetuando a leitura da entrada contida no arquivo `caso1.in` mas escrevendo a saída no arquivo `caso1.out` com adição do caminhamento em pré-ordem da árvore de pesquisa.

Entrada	**Entrada**
(mensagem 0) (mensagem 1) ... (mensagem $N-1$)	.-.. ..- ...- .-. . / . ... -.. --- .. / .-.. ..- ...- .-. . / . ... -.. --- .. --. ..-. --- / .--. --- ... ... --- / -. ..-. ..- ... / ... . -- .. .-. ..-. .-. .-.. ..- ...- .-. . / . ... -.. --- .. / .-.. ..- ...- .-. . / . ... -.. --- .. . .. / ... ..-. ..- / .--. .-. ..-. / --. ..-. --- / ...- --- .-.. -.. ..-. .-.
Saída	**Saída**
(mensagem 0 decodificada) (mensagem 1 decodificada) ... (mensagem $N-1$ decodificada) (primeiro símbolo) (primeiro código) (segundo símbolo) (segundo código) ... (último símbolo) (último código)	LIVRE ESTOU LIVRE ESTOU NAO POSSO MAIS SEGURAR LIVRE ESTOU LIVRE ESTOU EU SAI PRA NAO VOLTAR E . U .. F .- D - M -. G --

Tabela 2: À esquerda, um modelo genérico de entrada e saída do problema. À direita, um exemplo concreto, com quatro mensagens e o parâmetro *Exibir árvore*.

Entregáveis

Código-fonte. A implementação poderá ser feita utilizando as linguagens C ou C++. Não será permitido o uso da *Standard Library* do C++ ou de bibliotecas externas que implementem as estruturas de dados ou os algoritmos. **A implementação das estruturas e algoritmos utilizados neste trabalho deve ser sua**. Os códigos devem ser executáveis em um computador com *Linux*. Caso não possua um computador com *Linux*, teste seu trabalho em um dos computadores do laboratório de graduação do CRC[1].

[1] `https://crc.dcc.ufmg.br/infraestrutura/laboratorios/linux`

O código deve ser dividido em **pelo menos** três arquivos: ***main.c(.cpp)***, ***arvore.c(.cpp)*** e ***arvore.h***. O arquivo ***main.c(.cpp)*** deve conter as chamadas para as principais funções do seu programa. Os arquivos ***arvore.c(.cpp)*** e ***arvore.h*** devem conter, respectivamente, as implementações e as declarações da estrutura de árvore utilizada no trabalho. O arquivo `morse.txt` deve ser incluído no envio. A utilização de *Makefile*[2] é **obrigatória** para este trabalho.

O código-fonte terá diversas partes, incluindo porém não limitado a: leitura da codificação, leitura e interpretação do parâmetro, leitura da entrada, implementação da árvore, construção da árvore de pesquisa, caminhamento em pré-ordem na árvore, decodificação de caracteres e impressão da saída.

Aplique boas práticas de programação e organize seu código-fonte em arquivos, classes e funções de acordo com o significado de cada parte. A separação de responsabilidades é um dos princípios da engenharia de software: cada função deve realizar apenas uma tarefa e cada classe deve conter apenas métodos condizentes com sua semântica.

Documentação. A documentação de seu programa **deverá** estar em formato **PDF**, **seguir** o modelo de trabalhos acadêmicos da SBC (que pode ser encontrado online[3]) e ser **sucinta**. Deverá conter **todos** os seguintes tópicos:

- Cabeçalho. Título do trabalho, nome e número de matrícula do autor.
- Introdução. Apresentação do problema abordado e visão geral sobre o funcionamento do programa.
- Implementação. Descrição sobre a implementação do programa. Devem ser detalhados o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, compilador utilizado, bem como decisões tomadas relativas aos casos e detalhes que porventura estejam omissos no enunciado.
- Instruções de compilação e execução. Instruções de como compilar e executar o programa.
- Análise de complexidade. Estudo da complexidade de tempo e espaço do algoritmo de melhor e pior caso desenvolvido utilizando o formalismo da notação assintótica.
- Conclusão. Resumo do trabalho realizado, conclusões gerais sobre os resultados e eventuais dificuldades ou considerações sobre seu desenvolvimento.
- Bibliografia. Fontes consultadas para realização do trabalho.

O código-fonte e a documentação devem ser organizados como demonstrado pela árvore de diretórios na Figura 2. O diretório raiz deve ser nomeado de acordo seu **nome e último sobrenome**, separado por *underscore*, por exemplo, o trabalho de "Kristoff das Neves Björgman" seria entregue em um diretório chamado `kristoff_bjorgman`. Este diretório principal deverá conter um subdiretório chamado `src`, que por sua vez conterá o `morse.txt` e os códigos (`.cpp, .c, .h, .hpp`) na estrutura de diretórios desejada. A documentação **em formato PDF** deverá ser incluída no diretório raiz do trabalho. Evite o uso de caracteres especiais, acentos e espaços na nomeação de arquivos e diretórios.

O diretório deverá ser submetido em um único arquivo **'Nome_Sobrenome.zip'**, (onde Nome e Sobrenome seguem as mesmas diretrizes para o nome do diretório, explicado acima) através do Moodle da disciplina até as **23:59** do dia **4 de Julho de 2019**.

Considerações Finais

Algumas considerações finais importantes:

[2] `https://opensource.com/article/18/8/what-how-makefile`

[3] `http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros/878-modelosparapublicaodeartigos`

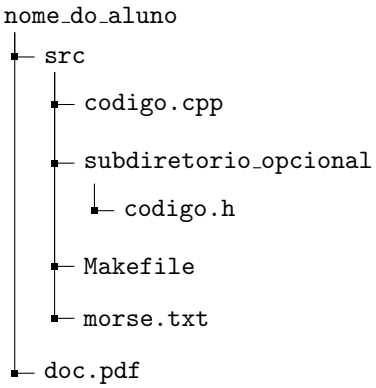

Figura 2: Estrutura de diretórios do entregável do TP3

- **Preste bastante atenção nos detalhes da especificação**. Cada detalhe ignorado acarretará em perda de pontos.
- O que será avaliado no trabalho:

  Boas práticas de programação: se o está código bem organizado e indentado, com comentários explicativos, possui variáveis com nomes intuitivos, modularizado, etc.

  Implementação correta dos algoritmos: se a árvore e o processo de decodificação foram implementados de forma correta e resolvem o problema aqui descrito.

  Conteúdo da documentação: se todo o conteúdo necessário está presente, reflete o que foi implementado e está escrito de forma coerente e coesa.
- Após submeter no Moodle seu arquivo **'.zip'**, faça o download dele e certifique-se que não está corrompido. Não será dada segunda chance de submissão para arquivos corrompidos.
- Em caso de dúvidas, **não hesite em perguntar** no Fórum de Discussão no Moodle ou procurar os monitores da disciplina – estamos aqui para ajudar!
- **PLÁGIO É CRIME**: caso utilize códigos disponíveis online ou em livros, **referencie** (inclua comentários no código fonte e descreva a situação na documentação). Trabalhos onde o plágio for identificado serão devidamente penalizados: o aluno terá seu trabalho anulado e as devidas providências administrativas serão tomadas. Discussões sobre o trabalho entre colegas são encorajadas, porém compartilhamento de código ou texto é plágio e as regras acima também se aplicam.
- Em caso de atraso na entrega, serão descontados $2^d - 1$ pontos, onde d é o número de dias (corridos) de atraso arredondado para cima.
- Comece o trabalho o mais cedo possível. **A data de entrega atual não poderá ser adiada por causa do fim do semestre**.

Bom trabalho!

Apêndices

A Dicas para a documentação

O objetivo desta seção é apresentar algumas dicas para auxiliar na redação da documentação do trabalho prático.

1. **Sobre *Screenshots***: ao incluir *screenshots* (imagens da tela) em sua documentação, evite utilizar o fundo escuro. Muitas pessoas preferem imprimir documentos para lê-los e imagens com fundo preto dificultam a impressão e visualização. Recomenda-se o uso de fundo branco com caracteres pretos, para *screenshots*. Evite incluir trechos de código e/ou pseudocódigos utilizando *screenshots*. Leia abaixo algumas dicas de como incluir código e pseudocódigo em seu texto.

2. **Sobre códigos e pseudocódigos**: Ao incluir este tipo de texto em sua documentação procure usar a ferramenta adequada para que a formatação fique a melhor possível. Existem várias ferramentas para LaTeX, como o `minted`[4], o `lstlisting`[5], e o `algorithm2e`[6] (para pseudocódigos), e algumas para Google Docs (`Code Blocks`[7], Code Pretty[8]).

3. **Sobre URLs e referências**: evite utilizar URLs da internet como referências. Geralmente URLs são incluídas como notas de rodapé. Para isto basta utilizar o comando `\footnote{\url{}}` no LaTeX, ou ativar a opção nota de rodapé[9] no Google Docs/MS Word.

4. **Evite o Ctrl+C/Ctrl+V**: encoraja-se a modularização de código, porém a documentação é única e só serve para um trabalho prático. Reuso de documentação é auto-plágio[10]!

[4] https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted
[5] https://www.overleaf.com/learn/latex/Code_listing
[6] https://en.wikibooks.org/wiki/LaTeX/Algorithms
[7] https://gsuite.google.com/marketplace/app/code_blocks/100740430168
[8] https://chrome.google.com/webstore/detail/code-pretty/igjbncgfgnfpbnifnnlcmjfbnidkndnh?hl=en
[9] https://support.office.com/en-ie/article/insert-footnotes-and-endnotes-61f3fb1a-4717-414c-9a8f-015a5f3ff4cb
[10] https://blog.scielo.org/blog/2013/11/11/etica-editorial-e-o-problema-do-autoplagio/#.XORgbdtKg5k