Documentação do Resgate Inesperado

Augusto Carvalho Porto Pereira

Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) Belo Horizonte – MG – Brazil

1.Introdução

Neste trabalho, precisamos orquestrar o retorno de várias espaçonaves repletas de robôs e recursos alienígenas que deverão chegar sãos e salvos em solo terrestre. Para isso, é necessário que a humanidade se comunique com essas naves de maneira criptografada para que os temíveis piratas espaciais não descubram e sabotem o plano de retorno ileso.

A solução desse problema está na criação de uma árvore de transliteração, criada com uma chave específica e super secreta, fora do conhecimento dos piratas, que é usada para codificar ou decodificar mensagens, estabelecendo uma comunicação de mão dupla entre as naves e as bases.

Na seção 2 deste documento são apresentados os detalhes de implementação da árvore binária de transliteração e seus métodos de criação, preenchimento e leitura. A terceira e última seção tratará da conclusão, bem como um resumo dos desafios enfrentados na construção do projeto.

2. Implementação

O sistema desenvolvido foi criado utilizando a estrutura de uma árvore binária, onde cada nó pertencente a essa árvore, se possuir filhos, terá sempre o filho à esquerda como menor do que a si mesmo e o filho à direita como maior do que a si mesmo. Para essa solução foram criadas as classes *ArvoreBinaria* e *Nodo*. além é claro do arquivo main para orquestrar o uso dessas classes.

Um objeto do tipo *Nodo* possui os campos *item*, do tipo char, *esquerda* e *direita,* ambos apontadores do tipo *Nodo.* Cada objeto representa um nó que contém seu caractere alfanumérico e um apontador para cada filho que, caso não exista, assumirá o valor nulo.

A *ArvoreBinaria* possui um elemento *raiz,* que é um *Nodo*. Para manipular a árvore, criando e lendo seus dados, existem os métodos públicos *Insere, CodificarCaractere* e *DecodificarMensagem,* que por sua vez chamando os respectivos métodos privados: *InsereRecursivo, CodificarCaractereRecursivo* e *DecodificarMensagemRecursivo*.

O método *Insere* é chamado para cada um dos 26 caracteres do alfabeto que são lidos no arquivo de entrada, na ordem pela qual eles foram informados. Caso sejam informados menos do que todos os arquivos de entrada, só estes serão populados na árvore, assim como se forem informados outros caracteres eles também serão inseridos na árvore. Por isso, para essa solução a complexidade de tempo e espaço é constante, e portanto, **O(1).** Para outra solução, com o número de elementos da árvore não fixado, sua complexidade no pior caso seria O(n).

O método *CodificarCaractere* recebe o caractere a ser codificado e gera uma sequência de valores semi-aleatórios representando esse caractere com base na cifra recebida. No pior caso, procurando o nó mais distante da raiz, a complexidade de tempo e espaço desse método é **O(h)**, sendo *h* o tamanho da árvore. O método possui complexidade de espaço também linear pois, por chamar o método privado recursivo, suas chamadas são empilhadas na pilha de execução.

O método DecodificarMensagem recebe a string com a chave para desencriptar cada caractere e retorna seu valor, também executando o método recursivo durante o processo, para uma complexidade **O(h)** de espaço e tempo assim como seu método irmão que realiza o processo inverso.

Já a complexidade do programa como um todo será dada pelo **número *n* de linhas no arquivo de entrada**. Como para todas as *n* linhas será executada uma codificação ou decodificação, e ambas têm complexidade de tempo O(h), a complexidade de tempo será **O(hn)**. Como no fluxo esperado desse sistema o valor de *h* é sempre igual a 26, a complexidade de tempo é então **O(n)**. Já a complexidade de espaço desse sistema é **O(h),** e utilizando o mesmo raciocínio da frase anterior, pode ser considerada simplesmente **O(1)**.

O sistema foi desenvolvido e testado utilizando um Windows Subsystem for Linux (WSL) que instancia a versão 18.04 do Ubuntu. O código foi totalmente implementado em C++ (além, é claro, do arquivo Makefile) utilizando o GCC C++ Compiler (g++). A execução dos testes foi realizada em uma máquina com 8GB de memória RAM e um processador i3 de 3.30Ghz.

3. Conclusão

Esse trabalho lidou com o problema de gerenciar uma árvore binária para construir um sistema de criptografia a distância e secreto para todos os que não possuem a chave de criptografia. Embora não seja o método de criptografia mais complexo já criado, ele se mostra uma solução simples e eficaz para o problema.

Com o aprendizado durante a construção dessa etapa, assim como em todas as outras etapas da missão, foi de suma importância para reforçar e fixar o conhecimento adquirido em sala de aula. E, graças a ele, foi possível garantir que em segurança os recursos cheguem no planeta e garantam a sobrevivência da humanidade, assim como o aprendizado se fixe nos alunos e abram novas portas e novos caminhos pela formação e pelo saber.

Referências

Chaimowicz, L. and Prates, R. (2020): Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.