Relatório de Algoritmos e Estrutura de Dados

Bruno Hermeto Guimarães[1]

[1]Pontifícia Universidade Católica de Minas Gerais (PUCMG)

Abstract. *This report presents the development of the practical work carried out in the discipline of Algorithms and Data Structures III, which, together with the theoretical classes, aimed to apply, in a practical way, several algorithms for indexing, external sorting, pattern matching, compression and cryptography. The implementations were developed in Java language using the Visual Studio Code environment, based on the materials and guidelines provided by the professors throughout the semester. The results obtained demonstrated the correct application of the algorithms and allowed a better understanding of the topics covered, highlighting the lessons learned throughout the semester.*

Resumo. *Este relatório apresenta o desenvolvimento do trabalho prático realizados na disciplina de Algoritmos e Estruturas de Dados III, que juntamente com suas aulas teóricas tiveram como objetivo aplicar, de forma prática, diversos algoritmos de indexação, ordenação externa, casamento de padrões, compressão e criptografia. As implementações foram desenvolvidas em linguagem Java utilizando o ambiente Visual Studio Code, com base nos materiais e nas orientações fornecidas pelos professores ao longo do semestre. Os resultados obtidos demonstraram a correta aplicação dos algoritmos e possibilitaram uma melhor compreensão dos temas abordados, com destaque para os aprendizados adquiridos durante a todo o semestre.*

1. Introdução

A disciplina de Algoritmos e Estruturas de Dados III tem como objetivo o estudo de algoritmos e estruturas voltados para o uso principalmente da memória secundária. Ao longo do semestre, foram explorados mais de 15 algoritmos e estruturas diferentes.

Nesse contexto, foi proposto um trabalho prático a ser desenvolvido ao longo de todo o período. Inicialmente, cada grupo deveria escolher uma base de dados com, no mínimo, 1000 registros, contendo campos diversos: números inteiros ou decimais, strings de tamanho fixo e variável, datas e listas com múltiplos valores. A base escolhida para este trabalho foi relacionada a pilotos de Fórmula 1, por ser um tema de interesse pessoal. Os registros da base incluíam campos de tipos inteiro, double, data, string de tamanho fixo, string de tamanho variável e uma lista de strings.

O trabalho foi dividido em quatro partes. O TP1 abordou a manipulação de arquivos sequenciais e algoritmos de ordenação externa. No TP2, foram exploradas técnicas de arquivos indexados e listas invertidas. O TP3 teve como foco os algoritmos de casamento de padrões e de compressão, e o TP4 tratou da implementação de métodos de criptografia. Ao longo dessas etapas, foi possível aplicar os conceitos aprendidos nas aulas teóricas e desenvolver soluções completas utilizando a linguagem Java.

2. Desenvolvimento

2.1. TP01 - Arquivo Sequencial e Ordenação Externa

O Trabalho Prático 01 (TP01) marcou a etapa inicial do projeto. Nessa fase, foi construída uma base de dados com informações sobre pilotos de Fórmula 1, utilizando dados coletados do Google e da Wikipédia. A base conta com mais de 1000 registros, contendo os seguintes campos: ID, títulos, participações, pódios, vitórias, pole positions e voltas mais rápidas (todos inteiros); pontos (tipo `double`); nome, país e função (strings de tamanho variável); data de nascimento (tipo data); sigla (string de tamanho fixo); e uma lista de equipes pelas quais o piloto correu (lista de strings).

Na segunda etapa, foi realizada a conversão da base de dados do formato CSV (separado por ponto e vírgula) para um arquivo binário com extensão `.db`. Para isso, utilizou-se o método `toByteArray()`, responsável por transformar um registro em um vetor de bytes. Durante esse processo, dois campos adicionais foram incorporados a cada registro: o tamanho do registro (para facilitar sua leitura) e uma *lápide*, indicando se o registro está válido ou foi logicamente excluído. A extração dos dados do arquivo binário foi feita com o método `fromByteArray()`, que realiza a operação inversa. O cabeçalho do arquivo armazena o ID do último registro inserido. A leitura e escrita nos arquivos binários foram feitas com a classe `RandomAccessFile`.

A etapa principal do TP01 consistiu na implementação das operações de CRUD (Create, Read, Update e Delete) para o arquivo sequencial. Para as buscas, foram implementados quatro métodos: o primeiro busca um piloto por ID e retorna o registro correspondente; o segundo realiza busca por nome; o terceiro e o quarto permitem buscas por sigla e por equipe, respectivamente, retornando uma lista com todos os pilotos compatíveis. O método de criação de um novo piloto recebe os dados inseridos, acessa o cabeçalho para obter o último ID usado, incrementa esse valor e adiciona o novo registro ao final do arquivo binário. Ao final da operação, o cabeçalho é atualizado com o novo ID. Para a exclusão, é feita uma busca pelo registro de ID correspondente e, em seguida, sua lápide é marcada como inválida, com isso o registro é logicamente excluído. O método de atualização localiza o registro a ser modificado e solicita os novos dados. Se o novo registro for de tamanho menor ou igual ao anterior, a atualização é feita diretamente no mesmo espaço. Caso o novo registro seja maior, o antigo é marcado como inválido e o novo é adicionado ao final do arquivo, preservando o mesmo ID. A implementação dos métodos de CRUD foi baseada nos vídeos e slides fornecidos pelos professores da disciplina.

Por fim, foi implementado o algoritmo de ordenação externa por intercalação balanceada, com o objetivo de reorganizar o arquivo por ID, considerando que exclusões e atualizações podem desordenar os registros. Durante a implementação, algumas dificuldades foram encontradas, e por isso o algoritmo não está completamente funcional. A implementação seguiu os vídeos explicativos e os exemplos de código disponibilizados no GitHub pelos professores da matéria.

2.2. TP02 - Arquivo Indexado e Lista Invertida

O Trabalho Prático 02 teve como objetivo a implementação de arquivos indexados utilizando Hash Extensível e um tipo de Árvore B, além da criação de duas listas invertidas com base em diferentes campos do registro.

Inicialmente, foi implementada a estrutura de Hash Extensível. O arquivo de índice correspondente armazenava pares compostos por ID e a posição do registro no arquivo de dados. A função hash utilizada foi $h(k) = k\%2^p$, em que p representa a profundidade atual do diretório. Sempre que o arquivo de dados sofria alterações, o índice hash também precisava ser atualizado para manter a consistência entre os arquivos.

Para o segundo índice, optou-se pela utilização de uma Árvore B+, cuja principal característica é a conexão lateral entre os nós folhas, o que facilita buscas sequenciais. A escolha pela B+ foi motivada pela menor complexidade de implementação em comparação com outras variações de árvores B. Assim como no Hash Extensível, o índice armazenava pares de ID e posições no arquivo de dados. No entanto, a implementação da árvore B+ apresentou dificuldades técnicas e, por isso, não ficou completamente funcional. Ambas as estruturas de índice foram baseadas nos vídeos explicativos e códigos disponibilizados no GitHub pelos professores da disciplina.

Após a implementação dos índices, foi desenvolvido um novo conjunto de operações CRUD compatível com os arquivos indexados. As operações de busca, inserção, exclusão e atualização se tornaram significativamente mais eficientes com o uso dos índices.

Por fim, foram criadas duas listas invertidas, que funcionam como dicionários para facilitar buscas textuais. Os campos escolhidos para indexação foram *nome* e *país*, por apresentarem uma grande diversidade de valores. Cada lista armazenava os termos únicos encontrados nesses campos, acompanhados das posições onde ocorriam nos registros. As buscas podiam ser realizadas de forma simultânea, permitindo consultas combinadas. A implementação das listas invertidas também seguiu as orientações fornecidas pelos vídeos e códigos disponibilizados pela disciplina.

2.3. TP03 - Casamento de Padrões e Compressão

O Trabalho Prático 03 teve como foco a implementação de dois algoritmos de casamento de padrões estudados em sala de aula, além da aplicação de dois algoritmos de compressão de dados: Huffman e LZW. O casamento de padrões foi aplicado em um campo específico dos registros, e os algoritmos de compressão foram utilizados para reduzir o tamanho dos arquivos de dados.

Na primeira parte do TP03, foram escolhidos os algoritmos KMP e Boyer-Moore para a realização do casamento de padrões. A escolha se deu por serem os primeiros algoritmos estudados na disciplina e por apresentarem estruturas lógicas bem compreensíveis. O campo utilizado para a busca foi o nome dos pilotos, por conter grande variedade e ser mais propenso a pesquisas textuais. O algoritmo KMP utiliza um vetor de prefixo (vetor de falha) que permite a realização de saltos durante as comparações, evitando comparações desnecessárias e otimizando o tempo de execução. Já o algoritmo Boyer-Moore combina duas heurísticas, o sufixo bom e o caractere ruim, que possibilitam os maiores saltos possíveis, tornando-o eficiente principalmente em buscas sobre grandes textos. Ambas as implementações foram realizadas com base nos vídeos explicativos da disciplina e nos exemplos de código em Python disponibilizados pelos professores.

Na segunda parte do TP03, foram implementados os algoritmos de compressão de Huffman e LZW. O algoritmo de Huffman constrói uma árvore binária com base na frequência dos caracteres presentes no texto. Caracteres mais frequentes recebem códigos

menores, o que contribui para uma compactação mais eficiente. A árvore gerada é salva em um arquivo auxiliar, utilizado posteriormente para a descompressão do conteúdo. O algoritmo LZW, por sua vez, baseia-se na construção de um dicionário dinâmico durante a leitura dos dados. À medida que novos prefixos são encontrados, o dicionário é atualizado, permitindo a substituição de sequências repetidas por códigos inteiros. Durante a descompressão, o mesmo dicionário é reconstruído para restaurar os dados originais. Ao final da execução desses dois algoritmos foi feita uma comparação entre ambos onde mostrou-se na maioria dos casos que o algoritmo LZW teve maior compactação e o Huffman teve melhor tempo de execução. As implementações seguiram os materiais teóricos da disciplina e os exemplos de código fornecidos no repositório da diciplina.

2.4. TP04 - Criptografia

No Trabalho Prático 04, foram implementados dois algoritmos de criptografia, um utilizando criptografia de chave simétrica, que deveríamos escolher um dos estudados em sala e o outro baseado em criptografia de chave assimétrica, que ficou determinado que seria o RSA.

Inicialmente, foi desenvolvido o algoritmo de criptografia com chave simétrica. Neste tipo de algoritmo, a mesma chave é utilizada tanto para criptografar quanto para descriptografar os dados. O método escolhido foi a cifra de Vigenère, um algoritmo clássico de substituição polialfabética. Ele funciona utilizando uma palavra-chave repetida em ciclos sobre o texto a ser criptografado. Para cada letra da mensagem, é aplicada uma rotação baseada na letra correspondente da chave, considerando a posição no alfabeto. Durante a descriptografia, o processo é revertido, subtraindo o valor da letra da chave para recuperar o caractere original.

Em seguida, foi implementado o algoritmo RSA, um sistema de criptografia assimétrica que utiliza um par de chaves: uma pública e uma privada. A segurança do RSA está baseada na dificuldade de fatorar grandes números primos. O algoritmo segue os seguintes passos principais: geração de dois números primos grandes p e q, cálculo do módulo $n = p \cdot q$, cálculo da função totiente $z = (p-1)(q-1)$, escolha de um expoente público e tal que $1 < e < z$ e $\gcd(e, z) = 1$, e finalmente o cálculo do expoente privado d, que é o inverso modular de e módulo z. A criptografia é realizada com a fórmula $C = M^e \% n$, e a descriptografia com $M = C^d \% n$, onde M representa o valor numérico da mensagem. Para adaptar esse processo a textos, o campo *nome* foi convertido para valores numéricos utilizando a codificação por bytes. Ambos os algoritmos foram implementados com base nos vídeos, slides e exemplos disponibilizados pelos professores da disciplina.

3. Conclusão

A realização dos quatro trabalhos práticos propostos na disciplina de Algoritmos e Estruturas de Dados III proporcionou uma ótima experiência sobre o uso de estruturas e algoritmos voltados ao tratamento de grandes volumes de dados em memória secundária. Ao longo das atividades, foram abordados tópicos fundamentais como manipulação de arquivos binários, estruturas de indexação, listas invertidas, algoritmos de casamento de padrões, algoritmos de compactação e técnicas de criptografia simétrica e assimétrica.

No TP01, foi possível compreender a estrutura de um arquivo sequencial, implementando um CRUD completo e aplicando ordenação externa com intercalação bal-

anceada. O TP02 permitiu explorar estratégias de indexação e recuperação de dados com ganho significativo de desempenho. Já no TP03, a aplicação de algoritmos de casamento de padrões e compressão evidenciou a importância de técnicas eficientes para manipulação e compactação de dados. Por fim, no TP04, foram aplicadas técnicas de criptografia que introduziram conceitos de segurança da informação diretamente no contexto da base de dados.

Apesar dos desafios enfrentados em algumas implementações, como na ordenação externa e na árvore B+, os trabalhos práticos foram fundamentais para consolidar os conteúdos teóricos estudados ao longo do semestre. Como resultado, obtivemos não apenas implementações funcionais em grande parte das propostas, mas também um aprofundamento significativo na compreensão dos algoritmos estudados.