Relatório do Trabalho Prático 01: Manipulação e Organização de Arquivos de Dados

Autor: Diego Henrique Marques Magalhães (22.2.8082) Disciplina: Algoritmos e Estruturas de Dados II Professor: Rafael Alexandre

1. Introdução

Este trabalho prático aborda a simulação de persistência de registros em um arquivo de dados (.DAT), com foco nas diferentes estratégias de organização e no gerenciamento de armazenamento em blocos de tamanho fixo. O objetivo principal é analisar e comparar, na prática, o impacto e a eficiência de três métodos distintos de alocação de registros:

1. Registros de Tamanho Fixo
2. Registros de Tamanho Variável (Contíguos)
3. Registros de Tamanho Variável (Espalhados)

O programa desenvolvido permite ao usuário gerar uma base de dados fictícia de alunos , definir o tamanho do bloco e escolher a estratégia de armazenamento , para então calcular e exibir estatísticas detalhadas sobre a eficiência e a ocupação do espaço em disco.

2. Estrutura e Decisões de Projeto

A solução foi desenvolvida na linguagem C, com foco na modularidade e clareza do código. O projeto foi dividido nos seguintes componentes principais:

- aluno.h / aluno.c: Define a estrutura de dados TAluno (conforme especificado na tabela do trabalho) e agrupa todas as funções relacionadas ao registro do aluno, como aluno() (para criação), imprime_aluno(), criar_base_alunos_ficticia() e as funções de cálculo de tamanho (tamanho_maximo_registro_aluno e tamanho_real_registro_aluno).
- simulacao.h / simulacao.c: Contém o núcleo da lógica do simulador. Este módulo define a struct TStats (para armazenar as métricas de saída) e implementa as três funções centrais de simulação (simular_escrita_fixa, simular_escrita_variavel_contigua, simular_escrita_variavel_espalhada), além da função exibir_estatisticas.
- main.c: Serve como o "controlador" do programa. É responsável por exibir o menu ao usuário, coletar os parâmetros de entrada (número de registros, tamanho do bloco, modo de simulação), inicializar os arquivos e coordenar a chamada das funções de simulação.

Essa separação de responsabilidades torna o código mais organizado, legível e fácil de manter.

3. Implementação das Estratégias de Armazenamento

O tratamento dos limites de bloco e o controle de bytes foram implementados da seguinte forma para cada estratégia:

3.1. Registros de Tamanho Fixo

- Função: simular_escrita_fixa()
- Lógica: O tamanho de cada registro é fixado pelo maior tamanho possível, obtido por tamanho_maximo_registro_aluno() (que usa sizeof(TAluno)).
- Controle de Bloco: Antes de escrever, a função verifica se o registro cabe no espaco_livre_bloco. Se não couber, o espaço restante é preenchido com o caractere # (usando a função preencher_bloco()), um novo bloco é iniciado (stats->num_blocos_total++), e o registro é movido integralmente para este novo bloco, garantindo a regra.

3.2. Registros de Tamanho Variável (Contíguos)

- Função: simular_escrita_variavel_contigua()
- Lógica: O tamanho de cada registro é calculado individualmente com tamanho_real_registro_aluno(a), que soma o tamanho dos campos fixos (int, float) com o tamanho real das strings (strlen() + 1).
- Controle de Bloco: A lógica é idêntica à de Tamanho Fixo, mas usando o tamanho real. Se o registro não couber no espaço restante, o bloco é preenchido com # e o registro é movido *integralmente* para o próximo bloco. Isso gera desperdício (fragmentação interna) no final dos blocos que não puderam acomodar o próximo registro.

3.3. Registros de Tamanho Variável (Espalhados)

- Função: simular_escrita_variavel_espalhada()
- Lógica: Esta é a estratégia mais complexa. Ela trata o registro como um fluxo de bytes (unsigned char* registro_bytes).
- Controle de Bloco: Um loop while controla a gravação do registro (enquanto bytes_gravados < tam_reg_total). A cada iteração, ele grava o máximo de bytes possível no espaco_livre_bloco. Se o bloco encher (espaco_livre_bloco == 0) e o registro ainda não tiver terminado, um novo bloco é alocado (stats->num_blocos_total++) e a escrita continua, cumprindo a regra de espalhamento. Esta estratégia minimiza o desperdício, pois não deixa espaços vazios no final dos blocos.

4. Cálculo de Estatísticas

Após a simulação, a função exibir_estatisticas() é chamada para apresentar os resultados.

- A struct TStats acumula os dados durante a simulação (ex: stats->bytes_dados_uteis, stats->num_blocos_total, stats->num_blocos_parciais).
- Cálculos Finais: Com base nesses dados, a função calcula as métricas finais:
    - Total de bytes (Arquivo): (double)stats->num_blocos_total * stats->tam_bloco
    - Total de bytes (Desperdício): total_bytes_arquivo - stats->bytes_dados_uteis
    - Percentual médio de ocupação: (stats->bytes_dados_uteis / total_bytes_arquivo) * 100.0
    - Eficiência de armazenamento: É o mesmo que o percentual médio de ocupação.
- Mapa de Ocupação: Um loop for imprime o mapa textual. Ele assume que todos os blocos, exceto o último, estão 100% cheios (uma simplificação para as estratégias de espalhamento) ou calcula a ocupação do último bloco com base nos bytes restantes.

5. Conclusão

A implementação do simulador permitiu observar o *trade-off* clássico entre simplicidade de implementação e eficiência de armazenamento.

- A estratégia de Tamanho Fixo é a mais simples de implementar, mas também a que mais desperdiça espaço, devido ao *padding* interno desnecessário em cada registro.
- A estratégia Variável Contígua melhora a eficiência ao usar o tamanho real, mas ainda sofre com a fragmentação interna no final dos blocos.
- A estratégia Variável Espalhada é a mais eficiente em termos de uso de espaço, aproximando-se de 100% de utilização (exceto pelo último bloco), mas sua implementação é significativamente mais complexa, exigindo um controle de bytes mais rigoroso.

O trabalho cumpre todos os objetivos propostos, fornecendo uma ferramenta prática para a visualização desses conceitos fundamentais de organização de arquivos.