Relatorio Sistema de Controle Sala de Aula

Pedro Henrique Farias Silveira, Enzo Alcantara Moura

September 2024

1 Introdução

O relatório a seguir documenta a implementação e o funcionamento de um sistema para controle de uma sala de aula com no máximo 100 (cem) alunos. Serão descritos detalhadamente a seguir os Tipos Abstratos de Dados (TADs) criados, bem como as funções implementadas para manipulá-los.

2 Estrutura do Sistema

2.1 TAD: Aluno

O tipo abstrato aluno faz o papel de um nó em uma estrutura no formato de lista encadeada, onde cada nó dessa lista representa um aluno em específico e armazena informações essencias a respeito dele, tais como:

- char name - Vetor de caracteres de tamanho MAXTAM (100) que contém o nome completo do aluno.
- int mat - Valor inteiro único que identifica o aluno, utilizado para representar a matrícula do mesmo.
- char curso[MAXTAM] - String que armazena o curso do aluno.
- int anoEntrada - Armazena o ano em que o aluno entrou.
- Aluno * prox - Referência para o próximo aluno da lista.
- int somaNotas - Armazena a soma de todas as notas do aluno.

2.2 TAD: ListaAlunos

O TAD ListaAluno é a estrutura da lista encadeada de alunos. Responsável por armazenar:

- Aluno * cabeca - Ponteiro para o nó do tipo aluno que representa o começo da lista (primeiro aluno).
- Aluno * fim - Ponteiro para o ultimo nó do tipo aluno inserido na lista.

2.3 TAD: Dia

Estrutura de lista encadeada que guarda informações sobre uma aula e a presença dos alunos.

- int dia - O dia em que a aula ocorreu.
- Presenca * cabeca - Ponteiro para o primeiro no da lista encadeada dia.

2.4 TAD: Presenca

É o nó da lista encadeada Dia.

- Aluno * aluno - Ponteiro para o aluno ao qual essa presença se refere.
- bool foi - Armazena 1 se o aluno estava presente e 0 se estava ausente.
- struct presenca * prox - Ponteiro para a próxima presença do dia.

2.5 TAD: NotaAluno

O tipo abstrato NotaAluno armazena as informações sobre as notas atribuídas ao alunos.

- int nota - Um valor do tipo float que representa a nota obtida pelo aluno.
- Aluno * aluno - Referência para o aluno associado a essa nota.

2.6 TAD: Avaliacao

O TAD Avaliação faz o papel de um nó de uma estrutura no formato de lista encadeada, onde cada nó dessa lista, nesse caso, cada Avaliação armazena informações a respeito de uma prova realizada.

- char nome[MAXLEN] - Vetor de caracteres que armazena o nome da avaliação. MAXLEN é definido como 100.
- float valortotal - Valor do tipo float que representa o valor total da avaliação (a nota máxima que pode ser alcançada por um aluno nessa avaliação).
- NotaAluno notas[MAXLEN] - Um vetor do tipo abstrato NotaAluno, contendo as notas de cada aluno para essa avaliação.
- struct avaliacao *prox - Referência para a próxima avaliação na lista.

2.7 TAD: ListaAvaliacoes

ListaAvaliacoes é a estrutura da lista encadeada de avaliações. Responsável por armazenar:

- Avaliação * cabeca - Referência a primeira avaliação da lista.

2.8 TAD: SalaDeAula

Estrutura utilizada para representar uma sala de aula (turma), e armazenar os alunos, as avaliações e as aulas.

- TabelaHashAluno * alunos - Tabela hash onde são armazenados os alunos.
- ListaAvaliacoes * avaliacoes - Lista encadeada com todas as avaliações cadastradas.
- Aulas aulas - Tipo Aulas é um vetor de ponteiros para Dia. Logo aulas armazena todas as aulas(presenças) cadastradas.

2.9 TabelaHashAluno

Tabela Hash utilizada para armazenar os alunos cadastrados possibilitando um acesso e uma inserção com complexidade O(1). O tratamento de colisões implementado foi o encadeamento externo.

- ListaAlunos * tabela[TABLE_SIZE] - Estrutura propriamente dita da tabela, um vetor de ponteiros para ListaAlunos.
- int indices_validos[TABLE_SIZE] - Vetor utilizado para armazenar quais posições da tabela já estão ocupadas. Utilizamos ele para quando formos gerar o relatório de alunos acessarmos diretamente apenas os índices da tabela que estão armazenando informação
- int contador_indices - Variável auxiliar utilizada para saber quantos índices de índices válidos teremos que percorrer para imprimir toda a tabela.

3 Funções

3.1 void gerar_matricula(Aluno * aluno)

Recebe um ponteiro para aluno e gera a matrícula do mesmo por meio de uma soma ponderada dos caracteres do nome do aluno e dos caracteres do curso multiplicada pelo ano de entrada do aluno.

3.2 int hash(int matricula)

Função hash de divisão aplicada à matrícula de um aluno. A hash de divisão foi escolhida pela sua simplicidade de implementação e seu desempenho satisfatório.

3.3 void insere_tabela_aluno(TabelaHashAluno * tabela, Aluno * aluno)

Recebe uma tabela de alunos e um ponteiro para aluno e insere o aluno recebido na tabela, tratando as colisões com encadeamento aberto.

3.4 Aluno * encontra_aluno(TabelaHashAluno * tabela)

Solicita ao usuário o nome, o curso e o ano de entrada do aluno que ele deseja encontrar, acessa esse aluno na tabela hash e retorna o ponteiro para ele.

3.5 void percorrer_tabela(TabelaHashAluno * tabela)

Percorre a tabela hash imprimindo as informações de matrícula, nome, curso e ano de entrada de cada aluno contido na tabela.

3.6 void verificaFaltas(Aluno * aluno)

Recebe um ponteiro para aluno e verifica se esse aluno atingiu a marca de dez (10) faltas, se sim imprime na tela um aviso dizendo que esse aluno foi reprovado.

3.7 void addFalta(Aluno * aluno, char c)

Recebe um ponteiro para aluno e um carácter (representa presença ou ausência) se o aluno estava ausente incrementa a quantidade de faltas do aluno.

3.8 void cadastraAluno(SalaDeAula * turma)

Solicita ao usuário as informações do aluno e insere esse novo aluno na tabela de alunos da turma. Se já existirem avaliações cadastradas na turma, solicita ao usuário que informe as notas dese novo aluno em cada uma delas. E se também já tiverem sido realizadas presenças solicita ao usuário que informe se o aluno estava presente (P) ou ausente (F) em cada um dos dias cadastrados, durante esse processo também é verificado o limite de faltas do aluno, se alcançado será impresso um aviso.

3.9 void cadastraAvaliacao(SalaDeAula * turma)

Solicita ao usuário o nome da avaliação, o seu valor total e cadastra essa nova avaliação na lista de avaliações da turma. Se já existirem alunos cadastrados na turma, solicita a nota de cada um deles nesta avaliação.

3.10 void realizaChamada(SalaDeAula * turma)

Solicita ao usuário a data em que a aula foi realizada, a presença de cada um dos alunos cadastrados na turma e insere esse Dia no vetor de dias da turma (Aulas aulas).

3.11 void relatorioAlunos(SalaDeAula * turma)

Solicita ao usuário por qual campo ele quer ordenar o relatório em seguida, pergunta qual algoritmo de ordenação ele deseja utilizar, são disponibilizadas duas

opções, merge sort (O(nlogn)) e selection sort ($O(n^2)$). Em seguida imprime o relatório de acordo com as preferências do usuário.

3.12 void relatorioNotas(SalaDeAula * turma)

Pergunta ao usuário o nome da avaliação a qual ele deseja gerar um relatório, caso o nome não for referente a nenhuma avaliação gera um aviso e volta ao menu, dando a opção do usuário chamar a função novamente. Imprime a maior nota, a menor e a nota média da avaliação, além de todas as notas tiradas pelos alunos, sem relacionar a nota ao aluno que a tirou.

3.13 void imprimirListasOrdenadas(TabelaHashAluno *tabela, int (*compara)(Aluno *, Aluno *), const char algoritmo)

Recebe a tabela hash e cria uma lista auxiliar e recebe a tabela ordenada nesta lista. O parâmetro algoritmo vai estabelecer qual algoritmo de ordenação será utilizado.

3.14 main

Na função main é implementado um menu onde o usuário interage com o programa, executamos as tarefas solicitadas pelo usuário fazendo chamadas as funções supracitadas necessárias para a execução do que foi solicitado pelo usuário através do menu.

4 Decisões

Algumas decisões tomadas ao longo do desenvolvimento do projeto valem a pena ser destacadas.

- **Uso de Tabela Hash para Alunos**: Optou-se por armazenar os alunos em uma tabela hash, utilizando a matrícula como chave de acesso. Essa decisão foi motivada pela necessidade de realizar inserções e buscas de forma eficiente, com complexidade média de O(1). A matrícula é gerada com base em atributos estáveis e praticamente únicos do aluno, como nome, curso e ano de entrada, minimizando colisões. Para tratar possíveis colisões, foi utilizado o encadeamento externo, mantendo a simplicidade e a eficácia do sistema.
- **Otimização da tabela hash com Vetor de Índices Válidos**: Para agilizar a geração de relatórios de alunos, foi implementado um vetor auxiliar que armazena apenas os índices válidos da tabela hash. Esse vetor permite acesso direto às posições ocupadas, evitando iterações desnecessárias sobre posições vazias e melhorando o desempenho geral da operação.

- **Cálculo da Matrícula**: A matrícula do aluno é calculada a partir de uma soma ponderada dos códigos ASCII do nome, curso e ano de entrada. Essa abordagem gera chaves diversificadas e reduz a probabilidade de colisões na tabela hash, garantindo um desempenho consistente.

- **Estruturas Encadeadas para Presenças e Avaliações**: Escolhemos listas encadeadas simples para gerenciar as presenças e as avaliações, uma vez que essas operações não exigem acesso direto aos elementos, mas sim consultas sequenciais para relatórios e verificações. Essa estrutura simplifica a implementação e mantém o sistema leve.

- **Encapsulamento das Estruturas na TAD SalaDeAula**: Para melhorar a organização e a legibilidade do código, todas as TADs relacionadas (alunos, presenças, avaliações) foram encapsuladas na TAD SalaDeAula. Essa abordagem centraliza o controle da sala de aula, facilitando a gestão de dados e a manutenção do sistema.

- **Uso de Algoritmos de Ordenação Específicos**: Implementamos dois algoritmos de ordenação para a geração de relatórios: o merge sort, com complexidade O(n log n), para operações mais eficientes em grandes conjuntos de dados; e o selection sort, com complexidade $O(n^2)$, para conjuntos menores ou onde a simplicidade do código era preferida. O usuário pode escolher o método de ordenação com base no tamanho e na natureza dos dados a serem ordenados.

- **Gestão de Avisos de Reprovação por Faltas**: A soma das faltas é armazenada diretamente na TAD Aluno para possibilitar verificações rápidas e avisos imediatos sobre a reprovação por excesso de faltas (10 ou mais). Essa escolha simplifica a lógica de validação e mantém o sistema responsivo em relação à frequência dos alunos.

- **Agrupamento de Dados por Aluno para Relatórios**: Notas e presenças são vinculadas diretamente aos alunos através de ponteiros nas TADs Avaliação e Presença. Essa decisão permite a geração de relatórios detalhados por aluno e por critério específico (ex: avaliação ou data), proporcionando uma visão mais integrada do desempenho e da frequência dos alunos.

- **Estrutura Modular**: O projeto foi subdividido em módulos de responsabilidade claramente definidas para facilitar a legibilidade do sistema e escalabilidade.