Relatório sobre o projeto - Trabalho de AEDs I

1) Estruturas criadas:

Notas – Tipos de dados:

Inteiro disciplina – Para identificação da disciplina ao qual a dada nota pertence;

Inteiro cod_avaliacao – Para identificação da avaliação ao qual a dada nota pertence. Necessário para encontrar as notas de uma avaliação desejada;

Float nota_aluno – Dado responsável por armazenar a nota da avalição do aluno, do tipo float para armezenar valores como 8,7 etc;

Ponteiro para o próximo – Ponteiro responsável pelo encadeamento das notas do aluno. Todas as notas de um aluno estarão encadeadas, mesmo sendo de disciplinas diferentes.

Aluno – Tipos de dados:

Inteiro matrícula – Armazena o identificador matrícula do aluno, cada aluno terá uma matrícula diferente. O sistema não permite matrículas iguais passarem, sendo assim, se uma matrícula igual for identificada ela será incrementada;

Caracteres nome – Armazena o nome do aluno;

Caracteres curso – Armazena o nome do curso do aluno, por exemplo, podemos ter um aluno no curso “Engenharia da Computação”;

Inteiro data_curso – Armazena a data de entrada no curso. Não há tratamento para correções, logo, deve sempre ser informado corretamente;

Inteiro faltas – Aqui será por onde teremos controle sobre alunos reprovados ou não. O sistema de chamada será apenas o número de faltas do aluno;

Float somatorio_notas – Para facilitar a ordenação pelo somatório das notas do aluno, foi criado esse tipo de dado. Ao cadastrar uma nota para o aluno esse valor será incrementado com o valor da nota cadastrada.

Ponteiro do tipo “Notas” – Aqui teremos o encadeamento das notas do aluno. É por aqui que teremos acesso a todas as notas cadastradas para um aluno específico;

Ponteiro para o próximo aluno – Responsável pelo encadeamento dos alunos cadastrados.

Avaliacao – Tipo de dados:

Caracteres nome_avaliacao – Armazena o nome da avaliação;

Float valor_avaliacao – Armazena o valor da avalição. É do tipo float para poder ter avaliações com notas que não sejam inteiras, mesmo sendo comum avaliações terem notas inteiras;

Inteiro cod_avaliacao – Armazena o código identificador da avaliação. Toda avaliação terá um código diferente. Não é permitido identificadores iguais, quando esse caso ocorrer, o processo é interrompido;

Inteiro disciplina – Armazena o código referente a qual disciplina a avaliação pertence. Não há tratamento, sendo necessário sempre informar corretamente;

Ponteiro para a próxima avaliação – Responsável pelo encadeamento das avaliações cadastradas. Todas as avaliações, independente de código ou disciplina ficarão encadeadas juntas.

Lista – Tipo de dado:

A lista apenas recebe a cabeça do encadeamento dos alunos. A cabeça do encadeamento é sempre o último aluno cadastrado.

Lista_Avs – Tipo de dado:

Recebe a cabeça do encadeamento das avaliações cadastradas. A cabeça do encadeamento é sempre a última avaliação cadastrada.

Lista_notas – Tipo de dado:

Recebe a cabeça do encadeamento das notas. A cabeça do encadeamento é sempre a última nota cadastrada. Utilizado apenas na ordenação das notas de uma avaliação. As notas serão encadeadas e inseridas nessa lista e depois serão organizadas por um Quicksort.

2) Principais decisões tomadas:

- A tabela hash nada mais é que um vetor de ponteiros do tipo Aluno. Esse vetor será inicializado com todas as suas posições apontando para NULL. A inserção nesse vetor só ocorrerá quando o índice dado pela "função_hash" apontar para um índice que aponte para NULL. Caso ocorra uma colisão a matrícula do aluno é incrementada e a função é chamada novamente de forma recursiva;

- O sistema de chamada, como já dito acima, apenas marca o número de faltas dos alunos. Desse modo, não é possível saber quais aulas e/ou dias o aluno faltou;

- Sempre que a matrícula de um aluno é igual a uma matrícula já cadastrada o sistema apenas incrementa a matrícula, mas o mesmo não ocorre com as avaliações e seus códigos. Por quê? Partindo de um ponto de vista de usuário, a matrícula cadastrada pode ser um valor qualquer, pois terão vários alunos e várias matrículas e ninguém irá saber todas as matrículas, apenas o sistema. Já os códigos das avaliações podem estar sendo cadastrados de forma específica, ou seja, um professor pode escolher o código dela para facilitar a geração de relatórios sobre a sua avaliação. Por isso quando ocorre a inserção de um mesmo código o sistema apenas termina;

- Todas as ordenações são feitas por Quicksorts. Fiz isso pois o Quicksort pode ter tanto a complexidade quadrática como a complexidade n(log(n)).

- Como podemos ver pelo código, há vários Quicksorts diferentes. Todos eles utilizam a mesma lógica, porém por terem que organizar tipos distintos, deixei-os como uma função separada para cada tipo de ordenação. O único Quicksort com uma lógica diferente é o "QuickSort", pois, quando o fiz, ainda não tinha definido com seria a TabelaHash e por isso não a utilizei nele.

Testes em relação a ordenação:

Como dito, meu algoritmo de complexidade quadrática e meu algoritmo de complexidade n(log(n)) serão ambos o mesmo, o Quicksort. Em "particiona", se meu pivô receber a menor nota ou a maior nota, ele se torna um algorítmo de complexidade quadrática. Temos então alguns resultados dos testes, sendo eles:

I) Quando utilizo poucos alunos cadastrados o tempo da demora não sofre grandes alterações. Ambos funcionam de forma bem rápida, porém o Quicksort com um bom pivô tem o resultado ligeiramente mais rápido.

II) Quando utilizo um número elevado de alunos o Quicksort que possui um bom pivô funciona normalmente e com um ótimo tempo, a demora é muito pequena. Porém quando utilizo um Quicksort com um pivô ruim, o tornando de complexidade quadrática, meu código quebra. Isso ocorreu todas as vezes em ambos os tipos de Quicksort.

III) Tendo falhado utilizando o Quicksort com complexidade quadrática, utilizei o Selection Sort. Comparando com os resultados do Quicksort com um bom pivô, não teve muita diferença no tempo. O Quicksort era ligeiramente mais rápido que o Selection sort quando havia por volta de 55 alunos.