Algoritmos e Estrutura de Dados II
Trabalho Prático 01

Lucas Pereira Monteiro[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

lucasmonteiro@dcc.ufmg.br

Resumo. *O objetivo deste trabalho é fazer uma aplicação que realize a montagem de uma grade de horários. Um exemplo prático: uma escola gostaria de alocar seus professores para o início do semestre letivo, logo, seria interessante que eles conseguissem fazer um gerenciamento da alocação e das colisões, para que não haja conflitos de horários, disciplinas e professores.*

1. Introdução

A manipulação de vetores (dinâmicos e estáticos) é um dos aspectos fundamentais (e básicos) em Ciência da Computação. Para se fazer um melhor uso das linguagens estruturadas[1], é importante utilizar os Tipos Abstratos de Dados, os quais "escondem" os detalhes de implementação. Outro aspecto interessante para se utilizar e que facilita bastante a manipulação de dados é o uso de Listas[2]

O objetivo desse trabalho é implementar alguns Tipos Abstratos de Dados (TAD) para a manipulação de vetores. Serão implementadas funções para leitura do vetor, pesquisa, inserção e impressão. Os TAD's darão suporte a um programa que tem como função principal fazer a alocação de disciplinas de uma escola. Também serão utilizadas listas encadeadas para ajudar na execução do programa.

Ao final do trabalho prático, espera-se praticar os conceitos básicos e necessários de programação utilizados no mesmo. São eles: manipulação de arquivos, manipulação de vetores, alocação dinâmica e listas encadeadas. Além da implementação e manipulação de TAD's, ponteiros e listas, os quais são extremamente necessários para um bom desenvolvimento em praticamente qualquer trabalho utilizando-se a linguagem C.

1.1. Funções, Procedimentos e Análise de Complexidade

CONSIDERAÇÃO AOS MONITORES: Como vocês podem perceber, criei muitas funções e só para listá-las aqui já foi requerido um grande espaço. Logo, não consegui colocar a função delas aqui, visto que o limite de páginas para a documentação, como especificado no fórum, foi de 6 páginas.

Por fim, todas as funções listadas abaixo são bem explicadas no código fonte, em todos os *.h*.

void FLHorarioVazia(TipoListaHorario *Lista); $O(1)$, pelo motivo que criar uma lista tem custo constante, não depende da entrada

[1]http://pt.wikipedia.org/wiki/Linguagem_de_programação

[2]http://www.ime.usp.br/ pf/algoritmos/aulas/lista.html

int LHorarioVazia(TipoListaHorario Lista); $O(1)$, pois o custo para comparar dois valores é constante

void InsereHorario(TipoCelulaHorario horario, TipoListaHorario *Lista); $O(1)$, pois se insere na lista com um custo constante, visto que a entrada é sempre de um elemento

void ImprimeHorario(TipoListaHorario Lista); $O(n)$, onde n é o tamanho de elementos da lista

void PreencheParametrosHorarios(TipoCelulaHorario *horario, char *linha); $O(1)$, o custo para preencher o tipo é constante

void ImprimeDiaSemana(int dia); $O(1)$, pois o número de itens passados na entrada é sempre constante (1)

void FLDisciplinaVazia(TipoListaDisciplina *Lista); $O(1)$, pelo motivo que criar uma lista tem custo constante, não depende da entrada

int LDisciplinaVazia(TipoListaDisciplina Lista); $O(1)$, pois o custo para comparar dois valores é constante

void InsereDisciplina(TipoCelulaDisciplina Disciplina, TipoListaDisciplina *Lista); $O(1)$, pois se insere na lista com um custo constante, visto que a entrada é sempre de um elemento

void ImprimeListaDisciplinas(TipoListaDisciplina listaDisc); $O(n)$, onde n é o tamanho de elementos da lista

TipoGrade AlocaVetorGrade(); $O(n.m)$, onde n é o número de linhas do arquivo e m é o número de classes

void PreencheVetorGrade(TipoGrade grade); $O(n.m)$, onde n é o número de linhas do arquivo e m é o número de classes

void ImprimeVetorGrade(TipoGrade grade); $O(n.m)$, onde n é o número de linhas e m é o número de colunas da matriz grade

void getOpt(int argc, char *argv[]); $O(1)$, pois o número de entradas é constantes

void AbreArquivos(char *paramEntrada, char *paramSaida); $O(1)$, pois o número de arquivos é constantes

void MostrarParametrosPassados(int argc, char *argv[]); $O(1)$, pois o número de entradas é constantes

FILE* AbreArquivo(char *arq); $O(1)$, pois o número de entradas é constantes

void FLColisaoDocentesVazia(TipoListaColisaoDocentes *Lista); $O(1)$, pelo motivo que criar uma lista tem custo constante, não depende da entrada

int LColisaoDocentesVazia(TipoListaColisaoDocentes Lista); $O(1)$, pois o custo para comparar dois valores é constante

void InsereColisaoDocentes(TipoCelulaColisaoDocentes docente, TipoListaColisaoDocentes *Lista); $O(1)$, pois se insere na lista com um custo constante, visto que a entrada é sempre de um elemento

void FLColisaoEspacoFisicoVazia(TipoListaColisaoEspacoFisico *Lista); $O(1)$, pelo motivo que criar uma lista tem custo constante, não depende da entrada

int LColisaoEspacoFisicoVazia(TipoListaColisaoEspacoFisico Lista); $O(1)$, pois o custo para comparar dois valores é constante

void InsereColisaoEspacoFisico(TipoCelulaColisaoEspacoFisico sala, TipoListaColisaoEspacoFisico *Lista); $O(1)$, pois se insere na lista com um custo constante, visto que a entrada é sempre de um elemento

void FLColisaoHorarioVazia(TipoListaColisaoHorario *Lista); $O(1)$, pelo motivo que criar uma lista tem custo constante, não depende da entrada

int LColisaoHorarioVazia(TipoListaColisaoHorario Lista); $O(1)$, pois o custo para comparar dois valores é constante

void InsereColisaoHorario(TipoCelulaColisaoHorario horario, TipoListaColisaoHorario *Lista); $O(1)$, pois se insere na lista com um custo constante, visto que a entrada é sempre de um elemento

int ExisteHorarioManha(int dia, int ordem, int turno, TipoGrade grade, TipoListaDisciplina *listaDisciplina, TipoListaColisaoHorario *listaColisaoHorario); $O(n.m)$, pois é necessário percorrer toda a matriz grade[][] para inspecionar os elementos

void ImprimeColisaoHorario(TipoListaColisaoHorario Lista); $O(n)$, onde n é o tamanho da lista

void ApontaParaCelulaVazia(TipoManha Manha, TipoManha Tarde, TipoNoite Noite, TipoCelulaClasse *vazia); $O(1)$, pois se faz somente uma atribuição

void PreecheMatrizTipoManha(TipoManha Manha, TipoGrade grade, TipoCelulaClasse *erro, TipoCelulaClasse *vazia); $O(n \exp 4)$, pois além de percorrer todas a matriz grade, que no caso terá custo $nexp2$, ainda se chama a função ExisteHorarioManha(), a qual tem complexidade $nexp2$. Portanto teremos um função com ordem de complexidade $nexp4$.

void PreecheMatrizTipoTarde(TipoTarde Tarde, TipoGrade grade, TipoCelulaClasse *erro, TipoCelulaClasse *vazia); $O(n \exp 4)$, pois além de percorrer todas a matriz grade, que no caso terá custo $nexp2$, ainda se chama a função ExisteHorarioManha(), a qual tem complexidade $nexp2$. Portanto teremos um função com ordem de complexidade $nexp4$.

void PreecheMatrizTipoNoite(TipoNoite Noite, TipoGrade grade, TipoCelulaClasse *erro, TipoCelulaClasse *vazia); $O(n \exp 4)$, pois além de percorrer todas a matriz grade, que no caso terá custo $nexp2$, ainda se chama a função ExisteHorarioManha(), a qual tem complexidade $nexp2$. Portanto teremos um função com ordem de complexidade $nexp4$.

1.2. Organização do código, Decisões de Implementação e Detalhes Técnicos

O código enviado foi divido em vários arquivos, para uma melhor organização do código. São eles: *colision_lists.h* , *general_lists.h*, *io.h* e *vetor_grade.h*. Além de todos os arquivos *.c*, referentes aos arquivos citados anteriormente, acrescidos do *main.c*.

A decisão se separação dos vários arquivos foi necessária para que o programa

ficasse mais organizado. Colision_lists trata funções criadas especialmente para tratar as colisões, General_lists contém as funções que criam as listas genéricas (usadas em todo o programa), Io cuida da parte de abertura e fechamento de arquivos, Vetor_grade é referente à manipulação do vetor grade que contém todas as classes que estão no arquivo e o arquivo Main faz o programa executar.

O compilador utilizado foi o Eclipse IDE for C/C++ Developers (Version: Juno Service Release 2) no sistema operacional Ubuntu 12.10.

Para executar o programa basta incializá-lo por linha de comando, como no exemplo seguinte:

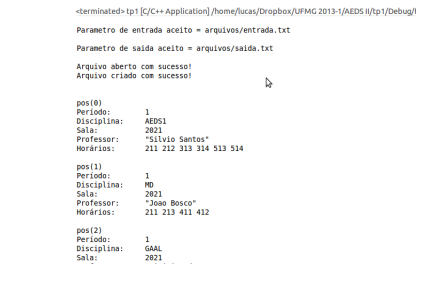
./exec -i entrada.txt -o saida.txt

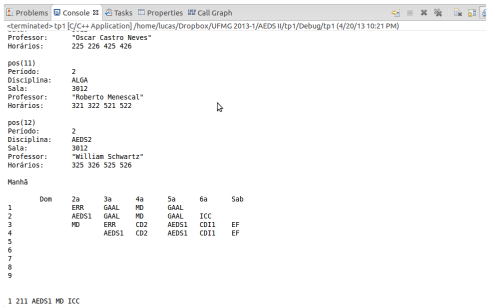
onde *exec* é o nome do executável, *entrada* contém o nome do arquivo de entrada, *saida* contém o nome do arquivo de saída, *-i e -o* são as flags utilizadas pela função getOpt, podendo também ser -I e -O.

2. Testes

Os testes foram realizados em um nootebook com processador *Intel Core i5, com 4Gb de memória RAM*. Porém, como não consegui terminar o programa, eles não tiveram os resultados esperados.

Seguem, um exemplo de inserção de classes e um exemplo de colisão simulado por mim.





3. Conclusão

O entendimento do problema proposto ocorreu sem maiores problemas. Porém tive grande dificuldade com a parte de implementação. Demorei muito tempo para conseguir

implementar todas as estruturas e listas necessárias e, no final, elas estavam funcionando, porém não tive mais tempo hábil para finalizar o projeto. Durante o início do projeto, defini algumas ideias que deveria seguir e fui até o fim com elas, mas chegando ao fim do projeto percebi que algumas delas não foram boas e me fizeram perder muito tempo tentando arrumar uma forma de finalizar o resto das funções.

Consegui inserir na grade, a qual continha a lista de classes, sem problemas. Também consegui detectar a colisão de horários, porém não consegui detectar os outros tipos de colisões, visto que já não me restava muito tempo para implementar o restante do trabalho.

Após a finalização do trabalho, consegui rever e me aprofundar mais nos conceitos já aprendidos em AEDS I, como alocação dinâmica e manipulação de arquivos. Outro resultado importante foi que aprendi a utilizar listas encadeadas e realocação, estruturas que considero muito importantes para o prosseguimento na disciplina, além de ter adquirido um bom conhecimento em manipulação de ponteiros. Portanto, imagino que os resultados também ficaram quase dentro do esperado, uma vez que pude praticar bastante os conceitos da linguagem C.

Por fim, acredito que o trabalho proposto cumpriu seu objetivo, de proporcionar ao aluno maior prática e aprendizagem da linguagem C.

Referências

Ziviani, N. (2004). *Projeto de Algoritmos com Implementações em C e Pascal*. Editora Thomson.