CCF 211 - Algoritmos e Estruturas de Dados I Trabalho Prático 01 - Agenda

Vinícius Mendes - 3881, João Roberto - 3883, Artur Papa - 3886

20 de março de 2021

1 Introdução

O trabalho em questão tem como objetivo abordar os conteúdos trabalhados na disciplina de Algoritmos e Estrutura de Dados I, em específico, como visto nas aulas teóricas, a utilização e criação de Tipo Abstrato de Dados(TAD) e lista encadeada. Um Tipo Abstrato de Dados (TAD) pode ser visto como um modelo matemático que encapsula um modelo de dados e um conjunto de procedimentos que atuam com exclusividade sobre os dados encapsulados. [1].

A priori, listas são estruturas muito flexíveis, porque podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda. Itens podem ser acessados, inseridos ou retirados de uma lista. [2]

2 Desenvolvimento

2.1 Dificuldades

Ao nos depararmos com os problemas relacionados ao desenvolvimento, tomamos algumas decisões que julgamos serem essenciais para a resolução da problemática. Buscando simplificar as implementações das listas encadeadas, separamos os TAD's em arquivos '.h' referentes ao TAD Compromisso e Agenda.

Além disso, outra dificuldade que tivemos durante o trabalho foi que usávamos sistemas operacionais diferentes, isso dificultou a implementação dos TADs, já que em diversos momentos os integrantes do grupo obtinham resultados diferentes, tudo devido as particularidades de cada sistema.

2.2 Lista Compromisso

Primeiramente, pode-se ressaltar que foi de grande necessidade a criação de uma lista para os compromissos, tendo em vista que seria de muita utilidade no momento de desenvolver as funções para o TAD Agenda.

Dessa forma, foi utilizada as funções básicas para a criação de uma lista, como a FLVazia, LRetira e LImprime [2], dito isso, foi desenvolvido uma nova forma de inserir os compromissos:

```
while(iter != NULL && (*celula)->compromisso->prioridade > aux->compromisso->prioridade){

    anterior=iter;
    iter=iter->pProx;
}if(iter= NULL){

    anterior->pProx = (*celula);

}
else{

    anterior->pProx->compromisso = (*celula)->compromisso;
    (*celula)->pProx = iter;
}
```

Figura 1: Função LInsere(insere compromissos)

Tendo como base a função acima, pode-se dizer que procuramos ordenar os compromissos a partir de cada item que estava sendo inserido. Buscamos utilizar diferentes "TCelulas"como itens auxliares nas realocações dentro da Lista de Compromissos, sendo elas: ***iter***(caminho), ***anterior***(referindo-se à celula anterior) e ***aux***(auxiliar). Realizamos a inserção comparando o novo compromisso, caso existam outros, com o item anterior e o item seguinte, buscando identificar a melhor posição deste.

2.3 Lista Agenda

Por conseguinte, a utilização da Lista Agenda tornou-se necessária tendo em vista nossa percepção em desenvolver o TAD Registro, organizando-a em instituições diferentes. Diante disso, ao observarmos a necessidade da implementação de uma lista de lista, a utilização de ponteiro duplo tornou-se primordial.

```
typedef struct Cell* pointer;
typedef struct Cell
{
    Agenda* agenda;
    struct Cell* prox;
}TCell;

typedef struct
{
    pointer fFirst;
    pointer fLast;
}TList;
```

Figura 2: Célula da Lista Agenda

2.4 Principais operações

2.4.1 Função Recupera Agenda

A proposta do TAD Agenda nos desafiou em diversos aspectos, pois visava a implementação de algumas funções mais complexas. Na função RecuperaAgenda, utilizamos alguns códigos desenvolvidos na função Insere Compromisso.

```
while (iter != NULL && celula->compromisso->dia != dia && celula->compromisso->mes != mes && celula->compromisso->ano != ano){

    anterior= iter;
    iter = iter->pProx;
    counter++;
}
if(iter = NULL){
    printf("Não ha essa data na agenda.\n");
}else{
    counter ++;
    printf("Nome do Professor: %s\n",pAgenda->nomeProf);
    printf("Ano: %d\n",pAgenda->ano);
    printf("Há %d compromissos apos essa data.",((retornaNCompromissos(&cabeca))-counter));
}
```

Figura 3: Parte da função Recupera Agenda

Dentro dessa função, foi realizada uma busca entre as células da lista encadeada (compromissos) com a finalidade de identificar a data passada como parâmetro. Ao identificá-la, devemos retornar os dados referentes à agenda do professor e o numero de compromissos que ele possui após a data informada. Utilizamos um contador para realizar essa cotagem.

2.4.2 Função Tem Conflito

A verificação contínua de diferentes parâmetros tornou a lógica por trás dessa operação um pouco desgastante.

```
while (((*compromisso)->dia == (*compromisso1)->dia) && ((*compromisso)->mes == (*compromisso1)->mes)
                    && ((*compromisso)->ano == (*compromisso1)->ano)){

    if(((*compromisso)->hora == (*compromisso1)->hora) && ((*compromisso)->minuto == (*compromisso1)->minuto)){

        printf("Há conflito entre os 2 compromissos");
        return 1;

    }else if (((*compromisso)->hora < (*compromisso1)->hora) && ((*compromisso)->minuto < (*compromisso1)->minuto)){
        if((horaFinal > (*compromisso1)->hora) && (minutoFinal > (*compromisso1)->minuto)){
            printf("Há conflito entre os 2 compromissos");
            return 1;
        }
    }else if (((*compromisso1)->hora < (*compromisso)->hora) && ((*compromisso1)->minuto < (*compromisso)->minuto)){
        if((horaFinal1 > (*compromisso)->hora) && (minutoFinal1 > (*compromisso)->minuto)){
            printf("Há conflito entre os 2 compromissos");
            return 1;
        }

    }else {

        printf("Não há conflito entre os compromissos");
        return 0;

    }
```

Figura 4: Parte da função Tem Conflito

A tarfa de abranger todos os casos possíveis, buscando trazer maior precisão no resultado, nos demandou bastante tempo além de gerar diversas dúvidas. A solução obtida foi a comparação entre cada item, interligando algumas funções com o objetivo de otimizar o códito, e realmente deixá-lo automatizado.

3 TAD Eventos e Registro

Logo após a implementação das funções desenvolvidas, foi decidido quais seriam os outros dois

TADs a serem efetivados, sendo eles o Evento e o Registro, haja vista que o primeiro tem como objetivo avisar aos professores caso tenha algum feriado, reforma no campi ou alguma ocasião especial, enquanto que o segundo foi usado para separar a agenda em instituições diferentes.

3.1 Eventos

A princípio vale ressaltar que o TAD Eventos foi criado no intuito de evitar conflitos entre o calendário feito pela universidade e pela agenda montada pelo professor. Outrossim, pode-se dizer que outra característica importante desse modelo de dados é que ele não é criado apenas pelos usuários da agenda, mas também podem ser modificados pelos administradores do sistema para adicionar algum evento.

3.1.1 Implementação

Como dito anteriormente o TAD Eventos fora implementado utilizando uma lista encadeada, as definições e funções forma feitas de forma análoga ao que foi visto em sala de aula e nos TADs Compromisso e Agenda. Com exceção das funções "void ChecarEventolista"e "void criasemanaEvento".

A função "void ChecarEventolista"checa se a lista de eventos foi inicializada, caso ela ainda não tenha sido iniciada, a função chama outra função a "void inicializaEventolista"que inicializa a lista. A ideia por trás dessa função é que como essa lista é acessada por todos os usuários não seria conveniente que ela fosse reinicializada diversas vezes sempre que alguém quisesse manipular a lista, portanto criamos essa função como uma ferramenta em que o próprio progama possa checar se a lista de eventos fora inicializada.

```
void ChecarEventolista(Clista* pListaEvento){
    if(pListaEvento->pPrimeiro->pProxC != NULL){
        inicializaEventolista(pListaEvento);
    }
}
```

Figura 5: Função para conferir se a lista foi inicializada

```
void checareventosmMes(Clista listaevento, int mes){
    Apontador iter;
    iter = listaevento.pPrimeiro;
    int contador = 1;

    while(iter->pProxC != NULL){
        if (iter->evento.mesinicial == mes){
            printf("********** Evento nº: %d **********",contador);
            printf("Descrição: %s \n", iter->evento.descricao);
            printf("Data do evento: %d / %d / %d \n",iter->evento.diainicial, iter->evento.mesinicial, iter->evento.anoinicial);
            printf("Hora do evento: %d : %d \n", iter->evento.horainicial, iter->evento.minutoinicial);
            printf("////////////////////////////////////");
            contador = contador + 1;
        }
        iter = iter->pProxC;
    }
}
```

Figura 6: Função que checa os eventos do mês

A função "void criasemanaEvento"serve para criar eventos com mais de um dia de duração, ela funciona pegando a data inicial e final do evento, com a restrição de que ambas as datas estejam no mesmo ano, e considerando todos os dias entre as datas como um evento.

```
void criasemanaEvento (Evento* Evento,int diainicial,int diafinal,int mesinicial,int mesfinal,int ano,int horainicial,int minutoinicial
,int horafinal,int minutofinal,int duracao,char *descricao){
    int iter = diainicial;
    while(iter != diafinal+1 && mesinicial!=mesfinal){
        criaEvento(Evento,iter,diafinal,mesinicial,mesfinal,ano,ano,horainicial,minutoinicial,horafinal,minutofinal,duracao,descricao);
        iter = iter+1;
        if(iter = 29 && (mesinicial = 2 && ano%4 == 0 && (ano%400 == 0 || ano%100 != 0))){
            iter = 0;
            mesinicial = mesinicial+1;
        }
        if(iter = 28 && (mesinicial = 2 && (mesinicial%4 !=0))){
            iter = 0;
            mesinicial = mesinicial+1;
        }
        if(iter==30 && (mesinicial==4 || mesinicial== 6 || mesinicial== 9 || mesinicial== 11)){
            iter = 0;
            mesinicial = mesinicial+1;
        }
        if(iter==31){
            iter = 0;
            mesinicial = mesinicial+1;
        }
    }
}
```

Figura 7: Função para eventos com mais de um dia de duração

3.2 Registro

O Registro tem como principal particularidade a união de diversas agendas, com o intuito de separá-las em grupos. Gerando assim uma nova capacidade para o sistema ser expandido além da UFV-Caf. Este TAD possui uma particularidade, a função Lê Arquivo, que é responsável por fazer a leitura de um arquivo e inicializar algumas funções, que tem como base o próprio TAD.

```
while (!feof(arq))
{
    fscanf(arq,"%d",&numprof);
    GeraReg(&ListRegistro);
    InsereReg(&ListRegistro, &registro);
    // printf("Insira o nome da Instituição: %s");
    // scanf("%s",&registro->instituicao);
    criaRegistro(&registro, Linha, 2021, "UFV");
    for (i = 0; i < numprof; i++)
    {
        fscanf(arq,"%d",&numCompromissos);
        fscanf(arq,"%d",&(fItem->idProf));
        fscanf(arq,"%d",&(fItem->nomeProf));
        fscanf(arq,"%d",&(fItem->ano));
        for (j = 0; j < numCompromissos; j++)
        {
            fscanf(arq,"%d",&prio);
            fscanf(arq,"%d",&day);
            fscanf(arq,"%d",&month);
            fscanf(arq,"%d",&year);
            fscanf(arq,"%d",&hour);
            fscanf(arq,"%d",&min);
            fscanf(arq,"%d",&minD);
            fscanf(arq,"%s",&desc);
            inicializaCompromisso(&(registro->CelulaAgenda->agenda->compromisso),prio,day,month,year,hour,min,minD,desc);
        }
    }
}
fclose(arq);
```

Figura 8: Função que realiza a leitura de Arquivos

Optamos por utilizar o "fscanf"pois identificamos maior precisão ao ler uma linha do documento, possibilitando assim a destinação de cada propriedade necessária.

4 Conclusão

Considerando as medidas e análises realizadas neste trabalho, inferimos que o trabalho em questão gerou diversos momentos de aprendizagem, tanto sob o âmbito da matéria, quanto ao compartilhamento e otimização do código. A que possuímos ao nomear parâmetros, váriaveis e structs era a de tornar o código mais limpo e claro, visando o entendimento de cada linha entre os integrantes.

Ao início do projeto, enfrentamos diversos desafios ao compartilhar os códigos que produzíamos até descobrirmos a ferramenta Git Hub Desktop, que tornou-se extremamente necessária para o desenvolvimento do projeto devido a sua facilidade em compartilhar atualizações no código.

Posto isso, é válido dizer que apesar das dificuldades na implementação do código, o grupo foi capaz de superar e corrigir diversos erros no desenvolvimento dos algoritmos. Haja vista que o trabalho foi executado conforme o planejado, sendo tratado tudo que foi pedido pelo professor e pelos monitores. Por fim, verificou-se a assertiva para o objetivo do projeto em criar um sistema automatizado de gerenciamento de tempo.

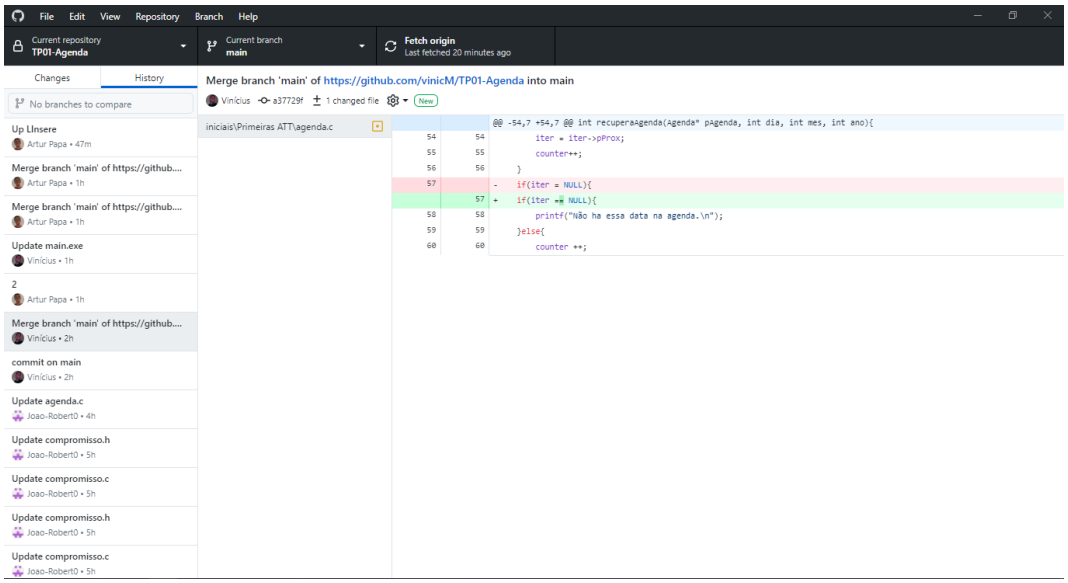

Figura 9: Git Hub Desktop

4.1 Imagens complementares

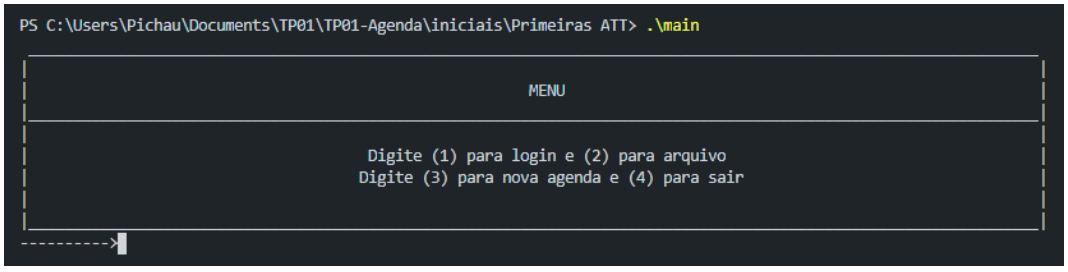

Figura 10: Menu 1 do sistema

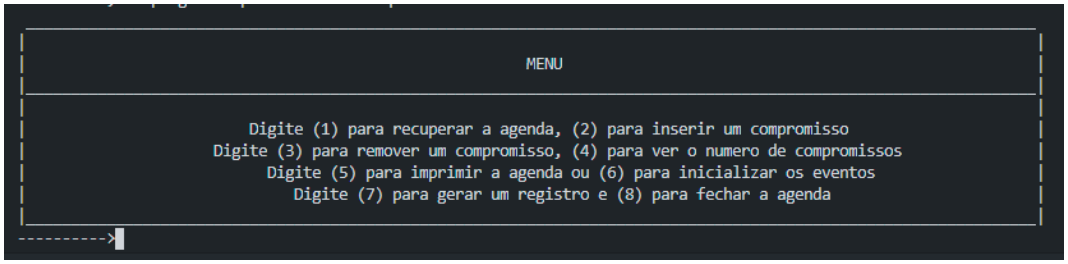

Figura 11: Menu 2 do sistema

```c
FILE *arqnova;
int l = 0;
arqnova = fopen("randidPROF.txt", "r");
while (!feof(arqnova))
{

    fscanf(arqnova, "%d", &testeidprof);
    while (testeidprof != idProf && l != 100)
    {
        fscanf(arq1, "%d", &testeidprof);
        l++;
    }
    if (testeidprof == idProf)
    {
        printf("o ID eh valido!");

        printf("\nDigite o ano: ");
        scanf("%d", &ano);
        printf("\nDigite seu nome: ");
        setbuf(stdin, 0);
        fgets(nome, 255, stdin);
        nome[strlen(nome) - 1] = '\0';
        criaAgenda(&pAgenda, idProf, ano, nome);
        printf("\nAgenda criada com sucesso!!!\n");
        menu1 = 1;
        break;
    }
    if (l == 100)
    {
        system("cls");
        system("clear");
        printf("--------------------------------Usurario nao encontrado--------------------------------\n");
        printf("---------------------Tente novamente Utilizando um usuario Valido---------------------\n");
```

Figura 12: Método de verificação de Login

Referências

[1] Tad:`http://www.inf.puc-rio.br/~coordicc/icc/TAD.pdf`.

[2] Nivio Ziviani. *Projeto de Algoritmos com implementações em Pascal e C*, volume 3. 2010.