

Primeiro Trabalho Prático de Algoritmos e Estruturas de Dados II

**Arthur Antunes Santos Silva, Dilvonei Lacerda
e Lucas Gonçalves Nojiri**

Professor: Rafael Sachetto Oliveira

Departamento de Ciências da Computação – Universidade Federal de São João del-Rei -
UFSJ – Campus Tancredo Neves

1- Introdução

Este trabalho teve como objetivo o aprendizado e fixação do conteúdo de AEDS 2, sobre o Tipo Abstrato de Dados e Listas. O tema do trabalho foi criar um algoritmo que auxiliasse os professores do DCOMP na organização de seus compromissos. Sendo assim, foi proposto que os alunos desenvolvessem um programa que funcionaria como uma agenda, permitindo os professores a organizarem seus afazeres, podendo adicionar seus compromissos, definirem seus tipos, alterarem seus horários e condições.

2- Implementação

2.1- Estruturas de Dados

Inicialmente foram criados os 5 tipos para o compromisso. O `tipo_de_compromisso` que define as categorias dos compromissos (Ex: AULA, ORIENTAÇÃO, REUNIÃO, EVENTO e COMPROMISSO PARTICULAR), `status_compromisso` onde o status desses compromissos estão contidos, em `TDATA` estão a data e horário e `TCOMPROMISSO` onde estão as definições dos compromissos. em seguida foi criado o tipo `TAGENDA` que contém as informações das agendas.

3- Funções Compromisso

3.1.1- int inicializa_compromisso(TCOMPROMISSO* comp)

Inicializa todos os compromissos de uma agenda, com os valores predefinidos para todo compromisso no instantes em que são criados, como por exemplo a variável `bool` adiável que inicialmente recebe `false`, e o status ser `A_SER_CUMPRIDO`, pois todo compromisso quando criado inicialmente deve ser cumprido e não pode ser adiado. Também definimos o identificador do compromisso igual a zero, para a cada compromisso podemos adicionar mais um, funciona de forma semelhante ao índice dos compromissos.

3.1.2- int altera_prioridade(TCOMPROMISSO* comp, int novaPri)

Essa função exibe a prioridade do compromisso atual (variando de 1 a 5), pela função `retorna_prioridade`, esta prioridade é importante para definirmos qual compromisso vai ser cancelado ou adiado em casos de conflito e a ordem de impressão. Em seguida pede para o usuário definir a nova prioridade do compromisso, caso seja digitado uma prioridade inexistente é exibido uma mensagem de erro, se for uma prioridade existente a mensagem de sucesso é exibida.

3.1.3- `int retorna_prioridade(TCOMPROMISSO comp)`

Recebe um compromisso como parâmetro e retorna a sua prioridade.

3.1.4- `bool Eadiavel(TCOMPROMISSO* c, bool adiável)`

Recebe como parâmetro um compromisso e uma variável bool, por exemplo passamos o compromisso `x` e `true`, se olharmos como uma pergunta séria, “o compromisso `x` é adiável?”. Caso um compromisso seja do tipo AULA ou EVENTO a função retorna “não podem ser adiados”, campo adiável vai ser falso, outros compromisso retornam o campo adiável como verdade, ou seja podem ser adiados.

3.1.5- `bool temConflito(TCOMPROMISSO comp1, TCOMPROMISSO comp2)`

Faz a verificação se existe conflito entre dois compromissos através de suas datas, horários e durações. Em compromissos no mesmo dia são comparadas as horas adicionando a duração em horas, isto acontece de duas formas: quando as horas são iguais ou quando está entre a hora inicial do compromisso e a hora final que seria a hora somada à duração em horas. No caso da mesma hora comparamos os minutos para definirmos de tem conflito. Para o tipo de compromisso evento existem algumas peculiaridades, pois a duração dos eventos podem ser em dias, compara-se a data que o evento começa e a duração em dias que seria data do final do evento, se existir algum outro compromisso entre esses dias irá retornar conflito igual a verdade, se não existir, será retornado conflito igual a falso.

3.1.6- `int atribuiStatus(TCOMPROMISSO*comp,status_compromiso status)`

Essa função recebe um compromisso e um status. Nela deve ser definido o status do compromisso, que dependendo do seu tipo ele deve ser adiado ou cancelado, dentro dessa função se faz o uso de outra, a `Eadiavel` para verificarmos se aquele tipo de compromisso pode ser adiado ou não. Se o tipo de compromisso não é adiável, a função automaticamente o define como cancelado, se o compromisso pode ser adiado, ele recebe o status passado no parâmetro.

3.1.7- `int retornaStatus(TCOMPROMISSO comp)`

Recebe um compromisso como parâmetro e retorna o status do compromisso.

3.1.8- void imprimecompromisso(TCOMPROMISSO *comp)

Imprime as informações associadas a um compromisso, seguindo a ordem dada: identificador, tipo, data, hora, duração, o tipo do compromisso, se é adiável e o status do compromisso. Os valores numéricos como o status e tipo do compromisso são impressos como cadeia de caracteres utilizando as funções `status_compromiso_string` e `tipo_de_compromisso_string` respectivamente. Para compreensão maior, a impressão da duração é realizada em dias(em caso de eventos), horas e minutos, esta convenção é feita utilizando a função `converte_dia_hora_minuto`.

```
ID Do Compromisso: 1
Tipo: AULA
Data : 20 / 3
Horario: 9:50
Duracao = Dias:0 / Horas: 1 e 40 Minutos
Nome do Compromisso: ISL
Nao Pode Ser Adiado
A_SER_CUMPRIDO
```

Figura 1 - Impressão de um compromisso.

3.1.9- char * tipo_de_compromisso_string(tipo_de_compromisso t)

Esta função de acordo ao tipo de compromisso, o enum tipo de compromisso, que vai de 1 a 5 , que são respectivamente ORIENTACAO , AULA, COMPROMISSO_PARTICULAR, EVENTO e REUNIÃO ela retorna uma dessas cadeias de caracteres, e teremos o próprio nome do tipo no lugar de um valor numérico.

3.2.0- char * status_compromiso_string(status_compromiso s)

Esta função de acordo ao status do compromisso, o enum status compromisso, que pode ser 0, 1 ou 2, que são respectivamente X X X ela retorna uma destas strings, e teremos o próprio Status no lugar de um valor numérico.

3.2.1- int* calcula_duracao_final(int duracao_minutos, TDATA inicial)

Recebe uma duração em minutos e um tipo TDATA, primeiramente faz a conversão da duração em minutos para dias(em caso de eventos) , horas e minutos com o uso da função `converte_dia_hora_minuto`, após isso, é feita a soma dos dias , horas e minutos, a fim de se obter o dia e o horário que um determinado compromisso vai acabar, em casos em que a o dia ultrapassa o dia 30, o mês vai para o próximo o dia recebe 1, o mesmo acontece com as horas e minutos, se as horas passam de 24, na variável do dia irá ser adicionado mais um, e as horas recebem 1(24 é igual à hora 00 por isso vai direto para o 1), para os minutos se é igual ou passa de 60, é adicionado mais um na hora, e os minutos zeram.

3.2.2-void converte_dia_hora_minuto(unsigned int duracao, int* duracao_dia_hora_minuto)

Essa função converte a duração passada em minutos para os valores de dias, horas e minutos e retorna um vetor com esses valores.

4- Funções Agenda

4.1.1- void cria_agenda(TAGENDA *agenda, int idProf, char* nome_professor, int ano)

Essa função recebe o endereço de agenda vazia, um id da agenda para o professor acessar essa agenda, um endereço de um vetor para o nome do professor e um ano. E a agenda vazia recebe todas essas informações. Toda agenda criada é definida com o tamanho igual a 0, pois está vazia(sem nenhum compromisso). Por fim é impresso o id da agenda, nome do professor e o ano da agenda e sempre após criar uma agenda é preciso inicializar os compromissos da mesma com a função inicializa_compromisso.

4.1.2- char recuperaAgenda(TAGENDA *agenda, TDATA data)

Recebe como parâmetro uma agenda e uma data e percorre toda agenda comparando se cada compromisso existente nesta agenda é depois desta data passada, cada vez que essa condição acontece adiciona mais um ao número de compromissos, após percorrer toda agenda retorna uma mensagem na tela com o nome do professor, ano, e o número de compromissos na agenda que são após a data passada, esta função tem complexidade n-1, pois preciso percorrer toda a agenda.

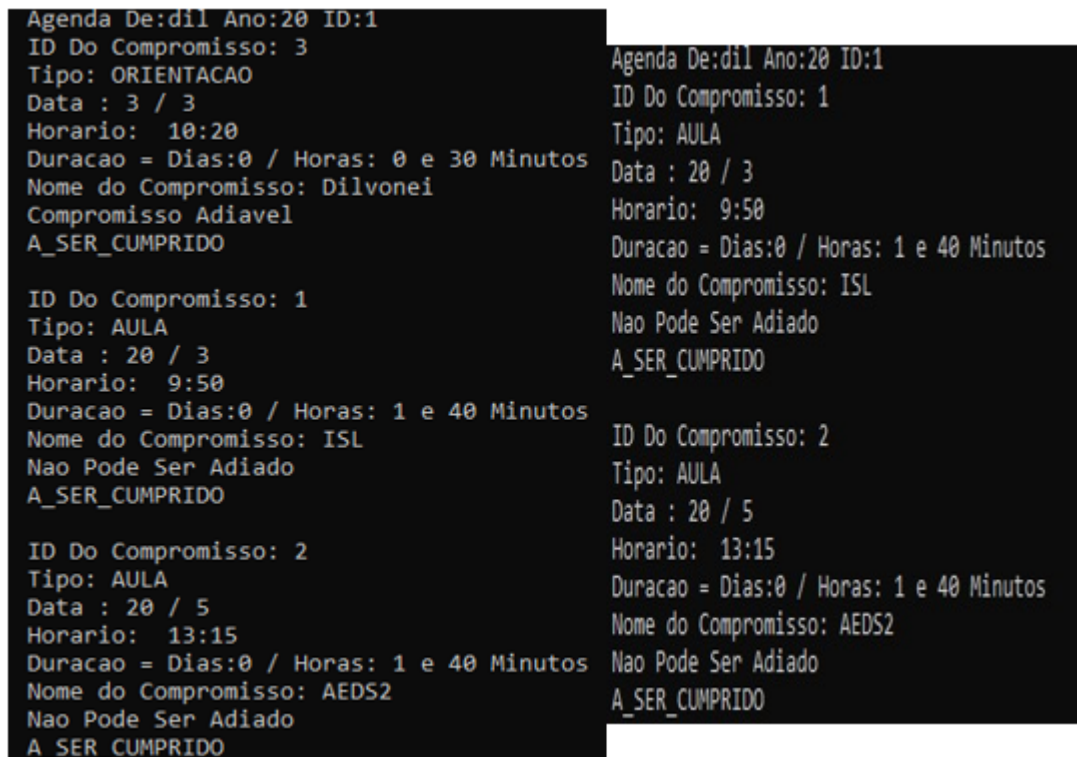
4.1.3- int insere_compromisso(TAGENDA *agenda, tipo_de_compromisso tipo, TDATA data, int duracao, char *nome)

Tem como parâmetros uma agenda e todas as informações de um compromisso(data, id, tipo, nome, duração e horário). As informações são inseridas no fim da agenda utilizando o tamanho atual da agenda. A variável prioridade recebe o tipo do compromisso, pois de acordo o tipo cada compromisso recebe um prioridade, isso foi necessário pois a prioridade em si é alterável, mas o tipo deve permanecer o mesmo. Ao fim é impresso uma mensagem de sucesso e o id do compromisso que é sempre igual ao tamanho mais um, ou seja, nesta implementação o id do compromisso é o índice mais um de cada compromisso.

4.1.4- int remove_compromisso(int id_compromisso, TAGENDA* agenda)

Essa função recebe como parâmetros uma agenda e o identificador de um compromisso, o usuário deve digitar o id do compromisso que deve ser removido. Caso seja digitado um id inválido, a função mostra quais são os ids existentes. O id do

compromisso -1 sempre será igual ao índice do mesmo. Por isso, quando um compromisso é removido e a agenda reorganizada, os seus id's também mudam, para que esta lógica continue sempre na agenda. Quando concluída a remoção uma mensagem de sucesso é imprimida.



The image shows two side-by-side terminal screenshots. The left screenshot displays the agenda state before a removal, listing three commitments (ID: 1, 2, 3). The right screenshot shows the state after the removal of the first commitment (ID: 1), where the remaining commitments (ID: 2, 3) have been re-indexed to (ID: 1, 2).

```
Agenda De:dil Ano:20 ID:1
ID Do Compromisso: 3
Tipo: ORIENTACAO
Data : 3 / 3
Horario: 10:20
Duracao = Dias:0 / Horas: 0 e 30 Minutos
Nome do Compromisso: Dilvonei
Compromisso Adiavel
A_SER_CUMPRIDO

ID Do Compromisso: 1
Tipo: AULA
Data : 20 / 3
Horario: 9:50
Duracao = Dias:0 / Horas: 1 e 40 Minutos
Nome do Compromisso: ISL
Nao Pode Ser Adiado
A_SER_CUMPRIDO

ID Do Compromisso: 2
Tipo: AULA
Data : 20 / 5
Horario: 13:15
Duracao = Dias:0 / Horas: 1 e 40 Minutos
Nome do Compromisso: AEDS2
Nao Pode Ser Adiado
A_SER_CUMPRIDO
```

```
Agenda De:dil Ano:20 ID:1
ID Do Compromisso: 1
Tipo: AULA
Data : 20 / 3
Horario: 9:50
Duracao = Dias:0 / Horas: 1 e 40 Minutos
Nome do Compromisso: ISL
Nao Pode Ser Adiado
A_SER_CUMPRIDO

ID Do Compromisso: 2
Tipo: AULA
Data : 20 / 5
Horario: 13:15
Duracao = Dias:0 / Horas: 1 e 40 Minutos
Nome do Compromisso: AEDS2
Nao Pode Ser Adiado
A_SER_CUMPRIDO
```

Figura 2 - impressão da agenda antes e depois de remover.

4.1.5- void imprime_agenda(TAGENDA *agenda)

Recebe como parâmetro uma agenda, o objetivo dessa função é imprimir todos os compromissos da agenda de acordo com a sua ordem cronológica(utilizando a função ordena_agenda), ou seja, a partir do instante de início. Caso haja um empate, deve ser considerado o compromisso com maior prioridade, se a prioridade é igual, o tipo do compromisso deve ser considerado na ordem de impressão: EVENTO, REUNIAO, AULA, ORIENTAÇÃO, COMPROMISSO PARTICULAR.

```
Agenda De:Sachetto Ano:2021 ID:1
ID Do Compromisso: 1
Tipo: AULA
Data : 6 / 6
Horario: 13:30
Duracao = Dias:0 / Horas: 0 e 50 Minutos
Nome do Compromisso: AEDS2
Nao Pode Ser Adiado
A_SER_CUMPRIDO

ID Do Compromisso: 2
Tipo: REUNIAO
Data : 10 / 6
Horario: 15:30
Duracao = Dias:0 / Horas: 0 e 25 Minutos
Nome do Compromisso: TP
Compromisso Adiavel
CANCELADO

ID Do Compromisso: 3
Tipo: ORIENTACAO
Data : 12 / 6
Horario: 17:30
Duracao = Dias:0 / Horas: 0 e 50 Minutos
Nome do Compromisso: Laboratorio
Compromisso Adiavel
A_SER_CUMPRIDO
```

Figura 3 - Impressão de uma agenda.

4.1.6- int resolve_conflicto(TAGENDA *agenda)

Esta função resolve todos os conflitos de horário encontrados pela função temConflito, toda vez que ela retorna que tem conflito entre dois compromissos as comparações, primeiramente de prioridade, serão feitas, aquele que tiver menor prioridade vai ser adiado ou cancelado utilizando a função atribuirStatus, porém se as prioridades são iguais, é resolvido pelo instante de início, se as duas condições são iguais, é preciso olhar o tipo de cada compromisso para atribuir um novo status ao compromisso, como cada tipo de status é equivalente é um valor, usamos um switch case a fim de reduzir o número de if's e else's.

```

Agenda De:Grupo Ano:2021 ID:1
ID Do Compromisso: 1
Tipo: ORIENTACAO
Data : 12 / 6
Horario: 13:0
Duracao = Dias:0 / Horas: 0 e 30 Minutos
Nome do Compromisso: Compromisso1
Compromisso Adiavel
ADIADO

ID Do Compromisso: 2
Tipo: AULA
Data : 12 / 6
Horario: 13:15
Duracao = Dias:0 / Horas: 0 e 50 Minutos
Nome do Compromisso: Compromisso2
Nao Pode Ser Adiado
CANCELADO

ID Do Compromisso: 3
Tipo: EVENTO
Data : 12 / 6
Horario: 13:10
Duracao = Dias:1 / Horas: 1 e 0 Minutos
Nome do Compromisso: Compromisso3
Nao Pode Ser Adiado
A_SER_CUMPRIDO

```

Figura 4 - Impressão de uma agenda com conflitos resolvidos.

4.1.7- int retorna_cancelamentos(TAGENDA* agenda)

Retorna o número de compromissos cancelados, esta função tem complexidade $n-1$, pois precisa percorrer toda a agenda para comparar os se os status de cada compromisso é igual a CANCELADO.

4.1.8- int retorna_adiamentos(TAGENDA* agenda)

Retorna o número de compromissos cancelados, esta função tem complexidade $n-1$, pois precisa percorrer toda a agenda para comparar se os status de cada compromisso é igual a ADIADO.

4.1.9- int retorna_compromissos(TAGENDA* agenda)

Recebe uma agenda e retorna o total de compromissos, esta função tem complexidade $n-1$, pois percorre toda agenda de tamanho n , porém ela poderia ser melhor somente retorna o tamanho atual da agenda que já é o número de compromissos daquela agenda.

4.2.0- void imprime_compromissos_adiados(TAGENDA* agenda)

Recebe um ponteiro de agenda e percorre a agenda. Se o status do compromisso estiver como ADIADO a função imprimecompromisso e chamada para imprimir os compromissos adiados.

4.2.1- void imprime_compromissos_cancelados(TAGENDA* agenda)

Recebe um ponteiro do tipo agenda e percorre todos os compromissos da agenda. Se o status estiver como CANCELADO a função imprimecompromisso é chamada para imprimir os compromissos cancelados.

4.2.2- void imprime_compromissos_acumprir(TAGENDA* agenda)

Recebe um ponteiro do tipo agenda e a percorre até o fim. Se o status estiver como A_SER_CUMPRIDO a função imprimecompromisso é chamada para imprimir os compromissos a serem cumpridos.

4.2.3- TAGENDA* ordena_agenda(TAGENDA *agenda)

Essa função recebe uma agenda como parâmetro e ordena os compromissos presentes na agenda seguindo a ordem cronológica dos compromissos. Exemplificando: se existem dois compromissos na agenda, o compromisso que se inicia primeiro vem no início da agenda, se forem iguais não importa para essa função, ela somente ordena pelo instante de início.

5- Função Main

Na função principal foi feito um loop contendo todas as opções de operações do algoritmo para ser usado como um menu, permitindo ao usuário digitar qual opção ele quer. Em cada condição foram chamadas as funções necessárias relativas a cada opção, no entanto na opção (2) Selecionar Agenda/Alternar Agenda, é somente onde é feita a busca das agendas existentes utilizando a variável tamanho_agenda, que começa com 0 e aumenta a cada agenda criada, ou seja, esta variável é a quantidade atual de agendas, detalhe que no momento da criação o id_agenda recebe essa variável mais um, sendo do mesmo modo dos compromissos, este id_agenda pode ser considerado o índice menos um da agenda.

6- Conclusão

As maiores dificuldades encontradas no TP, entre elas, as funções tem conflito e resolver conflitos, onde a tem conflitos deveria verificar entre os compromissos marcado se existiam conflitos e retorna true se existisse conflitos e false se não, se caso houvesse conflitos a função resolver conflitos iria solucioná-los atribuindo um status, sendo estes ADIADO ou CANCELADO de acordo com as regras dadas, inicialmente a ideia é simples, no entanto, devido ao número de comparações para encontrar um conflito foi complexo pensar em todas as comparações possíveis, sendo que para resolver uma das possibilidades de conflito, a função parava de resolver outra ou simplesmente aparecia uma nova possibilidade de conflito, para resolvermos isso separamos cada possibilidade(condição diferente) e resolvemos cada uma separadamente.

Ao final do trabalho concluiu-se que o tipo Abstrato de Dados e Listas são boas opções, pois facilita abstrair, pensar, como determinada função irá funcionar de acordo aos seus requerimentos e resolver problemas associados às mesmas.