

Detalhes de implementação do programa

Atenção: é recomendado que, antes da leitura desta documentação, faça-se a leitura do manual do usuário (arquivo *manual.txt*).

O programa é composto de quatro módulos (*Horarios.oz*, *Resolvedor.oz*, *Nucleo.oz* e *Adaptador.oz*), que serão descritos a seguir.

Módulo *Horarios.oz*

Este módulo provê uma interface texto simples com o usuário. Não há nada complexo em seu funcionamento, que pode ser resumido da seguinte maneira:

- Obtenção dos nomes dos arquivos de entrada e de saída (argumentos de linha de comando).
- Abertura do arquivo de entrada e obtenção dos dados.
- Obtenção da opção do usuário, que pode ser uma destas duas: gravar apenas a solução ótima no arquivo de saída; buscar uma solução qualquer, gravá-la no arquivo de saída e, a partir daí, repetir o processo de buscar uma solução melhor que a atual, gravando-a no arquivo de saída.
- Criação de um resolvedor (obtido a partir do módulo *Resolvedor.oz*) para o problema.
- Abertura do arquivo de saída.
- Inicialização de uma thread que lê a entrada padrão até que seja digitado sair<Enter> (quando isso acontece, a aplicação é encerrada, de forma adequada).
- Como o auxílio do resolvedor criado, é feita a resolução do problema, da forma indicada pelo usuário, e a gravação dos dados de saída.

Módulo *Resolvedor.oz*

Este módulo é bem mais complexo que o anterior. Seu objetivo é fornecer uma interface entre o módulo que efetivamente resolve o problema (*Nucleo.oz*) e o módulo *Horarios.oz*. Com isso, é possível fazer mais algumas otimizações na resolução do problema.

A interface deste módulo é o procedimento *CriaResolvedor*. Portanto, a explicação será feita em função dele.

Procedimento *CriaResolvedor*

Dados de entrada:

- O número de dias nos quais devem ser distribuídas as aulas.
- Um dicionário de turmas, indexado por átomos que identificam unicamente as turmas. Cada posição deste dicionário contém um registro *turma*(*prof:P exclMut:E aulas:Aulas dist:dist(min:Mi max:Ma) horVaria:B*), onde:
 - *P* é um átomo que identifica unicamente o professor desta turma.
 - *E* é uma lista de átomos identificando turmas. Cada turma que aparece nesta lista indica uma turma que tem exclusão mútua com esta.
 - *Aulas* é uma lista de aulas, cada uma sendo um registro *aula*(*dur:D hor:LH*), onde *D* é a duração da aula em minutos e *LH* é a lista de horários em que a aula pode começar (cada um na forma *Dia#Hora#Minuto*).

- Mi é a distância mínima (em dias) que pode haver entre duas aulas desta turma. Ma é análogo (para a distância máxima).
- B pode ser *true* (indicando que duas aulas desta turma não podem começar no mesmo horário do dia) ou *false*.
- Uma lista de listas de turmas. Cada uma das sub-listas indica um grupo de exclusão mútua.
- Uma lista de preferências, cada uma na forma $pref(prof:P\ indesejados:I\ peso:P)$, onde I é a lista de horários indesejados pelo professor, nesta preferência. Cada elemento da lista I tem a forma $Dia\#HI\#HF$, sendo HI a hora de início e HF a hora de fim do intervalo.
- Uma lista de restrições de horários dos professores, contendo elementos na forma $restrProf(prof:Prof\ impossiveis:LI)$, sendo LI a lista de intervalos (no mesmo formato utilizado nas preferências) em que o professor $Prof$ não pode dar aula.

Saída:

O procedimento *CriaResolvidor* devolve um registro na forma

$resolvidor(proximaSolucao:ProximaSolucao\ pausa:Pausa\ solucaoOtima:SolucaoOtima)$, onde:

- *ProximaSolucao* é um procedimento que busca uma solução melhor que a última encontrada e devolve a nova solução numa lista de um elemento (ou *nil*, caso não exista solução melhor que a última encontrada).
- *Pausa* é um procedimento que interrompe a busca atual (caso haja uma busca em execução).
- *SolucaoOtima* é um procedimento que busca a melhor solução possível para o problema e devolve esta solução numa lista de um elemento (ou *nil*, caso não exista solução).

Importante: após uma chamada ao procedimento *Pausa*, não é recomendado chamar novamente o procedimento *ProximaSolucao* (ou *SolucaoOtima*), pois, devido à implementação desses procedimentos, o comportamento resultante provavelmente não será o esperado.

Observe que não há problemas em chamar *SolucaoOtima* após chamadas a *ProximaSolucao*.

Funcionamento:

O procedimento *CriaResolvidor* funciona da seguinte maneira:

- É criada uma lista de objetos *Search.object*, obtida a partir do procedimento *CriaObjetosResolvidores**, que divide o problema dado em subproblemas independentes.
- Os subproblemas são resolvidos na ordem dada por essa lista de objetos.
- *NumObjResolvidores* é o número de subproblemas existentes.
- Existe uma célula (*ProblemaAtual*) que guarda o número do próximo subproblema que deve ser resolvido.
- Existe uma célula (*SolucaoAtual*) que guarda a melhor solução encontrada até o momento.

Procedimentos auxiliares:

- *UneSolucoes*: recebe uma lista de soluções para os subproblemas e devolve a solução correspondente do problema original.
- *AtualizaSolucao*: recebe uma solução do problema original e uma solução de um subproblema; atualiza a solução do problema original, levando em conta a solução do subproblema.

Procedimento ProximaSolucao:

- Quando é chamado pela primeira vez, este procedimento obtém a primeira solução de cada subproblema e utiliza *UneSolucoes* para gerar a primeira solução do problema original.
- Nas chamadas seguintes, os subproblemas são resolvidos, um por vez, da seguinte maneira:
 1. Se todos os subproblemas já foram resolvidos ($ProblemaAtual == NumObjResolvidores + 1$), a resposta é *nil*.
 2. Se não:
 - Encontra a próxima solução do próximo subproblema que deve ser resolvido.

- Se existir solução, a solução do problema original é atualizada e dada como resposta.
- Se não existir solução, incrementa-se *ProblemaAtual* e volta-se ao passo 1.

Procedimento *SolucaoOptima*: encontra a solução ótima de cada subproblema e utiliza *UneSolucoes* para gerar a solução ótima do problema original.

Procedimento *Pausa*: chama o método *stop* de cada objeto *Search.object* da lista *ObjetosResolvidores*.

* Procedimento *CriaObjetosResolvidores*:

- Encontra todos os professores cujas turmas não possuem exclusão mútua com nenhuma turma de nenhum outro professor. Os professores encontrados são os chamados professores independentes.
- Cada um desses professores (suas turmas, suas preferências e suas restrições de horários) forma um subproblema independente.
- Os professores restantes formam outro subproblema independente.

Observe que, se construirmos um grafo não-dirigido da seguinte maneira:

- Cada vértice representa um professor.
- Existe uma aresta entre o professor *A* e o professor *B* se e somente se existe alguma disciplina do professor *A* que possui exclusão mútua com alguma disciplina do professor *B*.

Então, cada componente conexo deste grafo forma um subproblema independente do problema original. Isso posto, fica claro que o procedimento *CriaObjetosResolvidores* apenas encontra os vértices isolados do grafo de professores. Um algoritmo mais forte (para encontrar todos os componentes conexos) não foi implementado apenas por limitações no tempo de desenvolvimento.

Módulos *Nucleo.oz* e *Adaptador.oz*

O módulo *Nucleo* contém a implementação da efetiva resolução do problema.

O principal procedimento (*ScriptHorarios*) recebe os dados de entrada de um problema e devolve um procedimento capaz de resolver o problema dado.

O outro procedimento implementado (*Ordem*) é importante como definição do conceito de solução ótima. O procedimento recebe duas soluções e cria um propagador para a restrição de que a segunda solução deve ser melhor que a primeira.

Procedimento *ScriptHorarios*

Este procedimento trabalha com unidades de tempo de 5 minutos (daqui para a frente, referir-nos-emos a essas unidades de tempo apenas como "unidades") e considera todos os dias do horário de aulas como um único intervalo de $(60/5)*24*NumDias = 288*NumDias$ unidades. Observe que 288 é o número de unidades existentes em um dia.

O primeiro passo executado pelo procedimento é transformar, com o auxílio do módulo *Adaptador*, os dados de entrada para um formato mais facilmente manipulável. Assim, o procedimento *AdaptaAoScript* (o único do módulo *Adaptador*) produz as seguintes estruturas de dados:

- *Aulas*: uma tupla de aulas, com todas as aulas da entrada, ou seja, todas as aulas de todas as turmas. A tupla é criada em ordem decrescente segundo uma avaliação que mede o grau de interferência de uma aula com as outras aulas do problema*. A posição das aulas nessa tupla será amplamente utilizada pelo programa. As aulas são armazenadas na forma *aula(turma:IdTurma dur:Dur hor:Horarios exclMut:ExclMut)*, onde:
 - *IdTurma* é o identificador único da turma a que essa aula pertence.
 - *Dur* é a duração desta aula, EM UNIDADES.
 - *Horarios* é a lista de horários possíveis, sendo cada horário apenas um número inteiro (o número

de unidades desde o início do primeiro dia). Por exemplo: se, nos dados de entrada, um dos horários possíveis era 2#8#0, então um dos números da lista *Horarios* será $(1*288)+(8*12)+(0*0) = 384$.

- *ExclMut* é a lista de aulas que possuem exclusão mútua com esta. A lista é composta apenas de números, cada um correspondendo a uma aula da tupla de aulas.
- *Turmas*: um dicionário, indexado pelos identificadores únicos das turmas, tendo, em cada posição, um registro na forma *turma(aulas:AulasDaTurma dist:Dist horVaria:HorVaria)*, onde:
 - *AulasDaTurma* são as aulas desta turma, na forma de uma lista de números, cada um correspondendo a uma aula da tupla *Aulas*.
 - *Dist* e *HorVaria* são idênticos aos campos correspondentes no dicionário de turmas dos dados de entrada.
- *Grupos* é uma lista de listas de números de aulas, sendo que cada sub-lista corresponde a um grupo de exclusão mútua (de aulas!).
- *Professores* é um dicionário, indexado pelos identificadores únicos dos professores, sendo que cada posição contém uma lista de números de aulas, correspondendo às aulas deste professor.
- *Preferencias* é uma lista com elementos na forma *pref(pref:Prof indesejados:Indesejados peso:Peso dificuldade:Dificuldade numDias:NumDias)*, onde:
 - *Prof* é o professor que tem esta preferência.
 - *Indesejados* é a lista dos intervalos indesejados nesta preferência (pela última vez, alerta-se para o fato de que a forma adotada para um horário é o número de unidades desde o início do primeiro dia).
 - *Peso* é o peso desta preferência.
 - *Dificuldade*** será explicado em detalhes mais adiante. Por enquanto, fique claro que *Dificuldade* indica a "dificuldade" (estimada pelo programa) de se satisfazer esta preferência. Quanto maior essa "dificuldade", menor a probabilidade (segundo a estimativa do programa) de que a preferência possa ser satisfeita.
 - *NumDias*: seja *IndAf* a lista de intervalos indesejados que afetam*** alguma aula. Então, *NumDias* é o número de dias que aparecem na lista *IndAf*. Esse número também será utilizado para estimar a dificuldade de se satisfazer uma preferência.

Importante: a lista de preferências é ordenada**** levando em conta os campos *Dificuldade*, *NumDias* e *Peso*. Preferências mais "fáceis" de serem cumpridas aparecem no começo da lista.

- *RestrProfessores* é a lista de restrições de horários dos professores. A forma de cada restrição é igual à forma de uma preferência, mas sem os campos "peso", "dificuldade" e "numDias" e com o campo "indesejados" substituído por "impossíveis".
- *MatrizExclMut* é uma matriz simétrica com *NumAulas* linhas e *NumAulas* colunas, sendo *NumAulas* o número de aulas do problema. Cada posição (*I*, *J*) contém *true* (se existe exclusão mútua entre a aula de número *I* e a aula de número *J*) ou *false* (caso contrário).

* Ordenação das aulas:

Na verdade, o procedimento *AdaptaAoScript* ordena as turmas. Assim, quando é criada a tupla *Aulas* (onde as aulas de uma mesma turma aparecem em posições adjacentes), a ordenação se reflete nas aulas.

O objetivo da ordenação de aulas é tornar a distribuição mais eficiente (veja, abaixo, a seção "Distribuição"), fazendo com que a seleção da variável mais restrita seja, na verdade, a seleção da variável mais "crítica" - dentre aquelas que têm domínio de tamanho mínimo.

Para permitir a ordenação das turmas, é necessária a criação das seguintes estruturas de dados:

- A lista *TodosOsGrupos*, representando todos os grupos de exclusão mútua, incluindo o conjunto das turmas de cada professor (pois cada um desses conjuntos é um grupo de exclusão mútua).
- O dicionário *DifTurmas*, com o seguinte significado:

- Cada chave é o identificador de uma turma.
- Um elemento, com chave *IdTurma*, é a avaliação da turma *IdTurma*, na forma *Dic#Num*, tal que:
 - *Dic* é um dicionário em que cada chave representa o tamanho de um grupo de exclusão mútua. Um elemento *Quant*, correspondente à chave *Tam*, indica que a turma *IdTurma* pertence a *Quant* grupos de exclusão mútua de tamanho *Tam*.
 - *Num* é a quantidade de turmas que aparecem na lista de exclusão mútua da turma *IdTurma*.

Para comparar duas turmas *T1* e *T2*, procede-se da seguinte maneira:

- Suponha que *Dic1#Num1* e *Dic2#Num2* são as avaliações, respectivamente, das turmas *T1* e *T2*.
- Se a maior chave de *Dic1* é maior que a maior chave de *Dic2* (ou se tais chaves são iguais, mas o elemento correspondente é maior em *Dic1*), então *T1* é mais “crítica” que *T2*. Raciocínio análogo vale para a determinação de que *T2* é mais “crítica”.
- Se o item acima não for suficiente, verificam-se a segunda maior chave de *Dic1* e a segunda maior chave de *Dic2*.
- Se *Dic1* e *Dic2* forem totalmente avaliados, sem resultados que permitam a comparação, então:
 - Se *Num1* > *Num2*, *T1* é considerada mais “crítica” que *T2*.
 - Caso contrário, *T2* é mais “crítica”.

*** Intervalo, de uma preferência, que afeta alguma aula:

O conceito é bem simples: o intervalo *I*, da preferência *Pref*, cujo professor é *Prof*, afeta alguma aula se e somente se existe alguma aula do professor *Prof* que, se for restrita a ter interseção não vazia com o intervalo *I*, terá sua lista de horários possíveis alterada.

** Campo "dificuldade" de uma preferência:

O campo dificuldade de uma preferência tem a forma *TamExclMax#NumAulasExclMax*, sendo que:

- *TamExclMax* é o tamanho do maior grupo de exclusão mútua que contém alguma aula afetada.
- *NumAulasExclMax* é o número de aulas afetadas que pertencem a algum grupo de exclusão mútua de tamanho *TamExclMax*.

**** Ordenação das preferências:

As preferências são ordenadas de acordo com os seguintes critérios:

- 1) Menor valor de *TamExclMax*.
- 2) (se 1 não for suficiente) Menor valor de *NumAulasExclMax*.
- 3) (se 1 e 2 não forem suficientes) Menor valor de *NumDias*.
- 4) (se 1, 2 e 3 não forem suficientes) MAIOR peso.

Observação: a idéia de utilizar o peso nas comparações não tem relação com a dificuldade de se cumprirem as preferências. O peso é utilizado apenas no sentido de que, se duas preferências são igualmente viáveis de serem cumpridas, é vantajoso que se tente cumprir primeiramente a preferência de maior peso.

Funcionamento do procedimento *ScriptHorarios*:

- Chama o procedimento *AdaptaAoScript*, para transformar os dados de entrada para um formato mais facilmente manipulável pelo script.
- Cria (e devolve) o procedimento que resolve o problema dado.

Procedimento devolvido por *ScriptHorarios*

Uma solução encontrada por este procedimento tem a forma *solucao(turmas:TurmasSolucao*

violadas:Violadas falhas:Falhas), onde:

- *TurmasSolucao* é um dicionário, indexado por turmas, contendo elementos na forma *aula(dia:Dia hor:Hor dur:Dur)*, onde:
 - *Dia* é o número do dia em que a aula deve ser dada.
 - *Hor* é o horário em que a aula deve ser dada, num formato humanamente legível (*Hora#Minuto*).
 - *Dur* é a duração da aula em minutos.
- *Violadas* é a lista de preferências violadas, cada uma na forma *violada(prof:Prof aulas:AulasQueViolam peso:Peso)*, onde:
 - *Prof* é o professor que tinha esta preferência.
 - *Peso* é o peso da preferência violada.
 - *AulasQueViolam* tem a forma *aula(turma:Turma dia:Dia hor:Hor dur:Dur)*, onde:
 - *Turma* é a turma desta aula.
 - *Dia*, *Hor* e *Dur* têm os mesmos formatos que aparecem nas aulas de *TurmasSolucao*.
- *Falhas* é a soma dos pesos das preferências violadas.

O procedimento atua da seguinte maneira:

PROPAGAÇÃO:

- Cria uma lista *NaoSatisfeitas*, em que cada posição corresponde a uma preferência (de acordo com a lista de preferências ordenadas), podendo conter 0 (preferência satisfeita) ou o peso da preferência (preferência não satisfeita).
- Cria um propagador para a restrição de que a soma dos elementos de *NaoSatisfeitas* é o custo (número de falhas) da solução.
- Cria uma tupla (*Inicios*) de horários de início para as aulas (na ordem dada pela tupla *Aulas*).
- Cria uma tupla de dias para as aulas, na ordem dada pela tupla *Aulas*. Os valores desta tupla são calculados a partir da tupla *Inicios*, ou seja, a tupla *Dias* não é necessária (é uma tupla "auxiliar").
- Restringe os horários das aulas aos horários dados pela entrada.
- Restringe os horários das aulas de acordo com as restrições de horários dos professores.
- Cria propagadores para restringir as distâncias, em dias, entre as aulas de uma mesma turma.
- Cria propagadores para a eliminação de simetrias: se duas aulas (*A1* e *A2*) de uma mesma turma são idênticas (possuem a mesma lista de horários possíveis e a mesma duração), pode-se criar a restrição de que *A1* deve ser dada antes de *A2*. Isso melhora o desempenho do programa.
- Cria propagadores para os grupos de exclusão mútua. O conjunto das aulas de um professor também é considerado um grupo de exclusão mútua.
- Cria propagadores para as outras restrições de exclusão mútua entre aulas.
- Cria propagadores para os casos em que duas aulas de uma mesma turma não podem ser dadas no mesmo horário do dia.

DISTRIBUIÇÃO:

- 1) Faz-se a distribuição nas preferências, em ordem crescente de "viabilidade". Para cada preferência, exploram-se primeiramente os casos em que a preferência é satisfeita. Os objetivos dessa forma de distribuição, tendo em vista que a busca é feita em profundidade, são:
 - Fazer com que as soluções "boas" sejam encontradas primeiro. Em particular, as soluções perfeitas (ou seja: que têm todas as preferências satisfeitas), caso existam, são as primeiras a aparecerem na busca. Com isso, a busca terminará rapidamente nos casos de problemas que têm ao menos uma solução perfeita.
 - Caso esteja correta a estimativa feita pelo programa para a "viabilidade" das preferências, a ordenação das preferências terá um papel crucial no desempenho, pois os casos "mais viáveis" serão explorados antes.

2) Faz-se a distribuição (obrigatória, para que o problema possa ser resolvido) nos horários de início, utilizando-se uma estratégia que consiste em escolher primeiramente a variável (no caso, a aula) mais restrita* (com menos horários de início possíveis) e, em seguida, escolher o valor menos restritivo** para esta variável (ou seja: tentar primeiro o valor menos restritivo e depois os outros valores).

* Observe que, como as aulas estão ordenadas, será escolhida a variável mais “crítica”, dentre as que têm domínio de tamanho mínimo. Este fato também tem grande influência no desempenho.

** Seria muito custoso determinar com exatidão o efeito (sobre as outras variáveis) da escolha de determinado valor para uma variável $A1$. Portanto, utiliza-se uma estratégia mais simples:

- Seja $IndetExclMut$ a lista das variáveis ainda não determinadas (domínio estritamente maior que 1) que possuem exclusão mútua com $A1$.
- Para cada valor $V1$ do domínio de $A1$, cria-se um dicionário tal que: cada chave Tam representa o tamanho do domínio de ao menos uma variável de $IndetExclMut$, caso $A1$ recebesse o valor $V1$ e fossem aplicadas as restrições de exclusão mútua; o elemento correspondente representa a quantidade de variáveis de $IndetExclMut$ que teriam domínio de tamanho Tam .
- Para fazer a comparação entre dois valores $V1$ e $V2$, procede-se da seguinte maneira:
 - Sejam $Dic1$ e $Dic2$, respectivamente, os dicionários associados a $V1$ e $V2$ no passo anterior.
 - Se a menor chave de $Dic1$ é maior que a menor chave de $Dic2$ (ou se as chaves são iguais, mas o elemento correspondente é menor em $Dic1$), então $V1$ é menos restritivo que $V2$. Raciocínio análogo aplica-se para a verificação de que $V2$ é menos restritivo que $V1$.
 - Se o passo acima não for suficiente, analisa-se a segunda menor chave de $Dic1$ e a segunda menor chave de $Dic2$. Se necessário, os dicionários são analisados por completo. Se não houver valor menos restritivo, seleciona-se o menor valor.