Lógica para Programação

Projecto

2022-2023

Horários – pesquisas em Prolog

Conteúdo

Um belo dia, ouves dizer que a base de dados relativa à ocupação de salas do IST-Tagus foi atacada por um *hacker*. Na tentativa de ajudar, estudas as estruturas de dados em causa (Secção 1), crias um programa em Prolog (primeiras linhas na Secção 2) e resolves o problema. Tornas-te um herói, mas, como sabes, com grandes poderes vêm grandes responsabilidades, e começam a chegar-te mais pedidos de ajuda (Secção 3). Claro está, dás o teu melhor! Sobre as condições de realização do projecto, sua avaliação e recomendações, vê as Secções 4 e 5.

1 Estruturas de dados

Existem dois ficheiros – dados.pl e keywords.pl – que constituem parte de uma versão livremente modificada de uma base de conhecimento gentilmente cedida pela Área Académica e Gestão do Edifício. O ficheiro dados.pl contém factos sobre eventos, turnos associados aos eventos e horários dos eventos, definidos como se segue:

Um evento, `evento(ID, NomeDisciplina, Tipologia, NumAlunos, Sala)` caracteriza-se por:

- Um identificador;
- O nome da disciplina associada ao evento;
- A tipologia do evento (seminário, teórica, etc.);
- O número de alunos associado ao evento;
- A sala em que ocorre o evento.

Um evento tem associados um ou mais turnos, `turno(ID, SiglaCurso, Ano, NomeTurma)`, caracterizados por:

- Um identificador (o ID do evento associado);
- A sigla do curso a que diz respeito o evento;
- O ano em que a disciplina é oferecida no curso;
- O "nome" do turno.

Um evento tem ainda associado um horário, `horario(ID, DiaSemana, HoraInicio, HoraFim, Duracao, Periodo)`, caracterizado por:

- Um identificador (o ID do evento associado);
- O dia da semana em que ocorre o evento;
- As horas de início e fim do evento[1];
- A duração do evento (sim, podia ser deduzida dos valores anteriores);
- O período em que ocorre o evento.

A título de ilustração, os factos que se seguem indicam que o evento 10 diz respeito a um laboratório da disciplina de 'Sistemas Digitais', com 18 alunos e que decorre na sala 1-62. Tem lugar sextas-feiras, entre as 8h e as 10h, tendo, por isso, uma duração de duas horas, e decorre no p2. Este evento é do primeiro ano da LEE, turmas lee0101 e lee0102.

[1] Assume-se que todos os eventos ocorrem à hora ou à hora e meia. Assim, para facilitar os cálculos, usar-se-à 12.5 para indicar as 12h30 (por exemplo, ao somar 1h30 às 12h30, estaremos a somar 1.5 a 12.5, obtendo-se 14 (14h)).

```
evento(10, 'sistemas digitais', laboratorial, 18, '1-62').
horario(10, sexta-feira, 8.0, 10.0, 2.0, p2).
turno(10, lee, 1, lee0102).
turno(10, lee, 1, lee0101).
```

No ficheiro "keywords.pl" encontram-se keywords que serão úteis:

```
salas(grandesAnfiteatros, ['a1', 'a2']).
...
salas(videoConf, ['0-19', '0-13']).
...
licenciaturas(tagus,['lee', 'legi', 'leic-t', 'leti']).
mestrados(tagus,['mbmrp', 'mee', 'megi', 'meic-t', 'meti']).
```

2 O programa em Prolog

O ficheiro em Prolog (extensão pl), que será usado no projecto, deverá ter as seguintes linhas iniciais:

```
% Numero e o nome do aluno
:- set_prolog_flag(answer_write_options,[max_depth(0)]). % para listas completas
:- ['dados.pl'], ['keywords.pl']. % ficheiros a importar.

/*
Codigo
*/
```

3 Predicados a implementar

~~3.1 Qualidade dos dados~~

O primeiro pedido de ajuda vem da Secretaria: pedem-te para os ajudares a encontrar eventos problemáticos; em especial pedem-te para identificar/encontrar:

- Eventos sem salas;
- Eventos sem salas, dado um dia da semana;
- Eventos sem salas, dado um período;

Arregaças as mangas e abraças o desafio com entusiasmo.

Sabendo que os eventos sem salas são identificados por terem a palavra 'semSala' no campo relativo à sala, implementas os predicados `eventosSemSalas/1`, `eventosSemSalasDiaSemana/2` e `eventosSemSalasPeriodo/2`, tais que (respectivamente):

– `eventosSemSalas(EventosSemSala)` é verdade se `EventosSemSala` é uma lista, ordenada e sem elementos repetidos, de IDs de eventos sem sala;
– `eventosSemSalasDiaSemana(DiaDaSemana, EventosSemSala)` é verdade se `EventosSemSala` é uma lista, ordenada e sem elementos repetidos, de IDs de eventos sem sala que decorrem em `DiaDaSemana` (doravante `segunda-feira, terca-feira, quarta-feira, quinta-feira, sexta-feira, sabado`);
– `eventosSemSalasPeriodo(ListaPeriodos, EventosSemSala)` é verdade se `ListaPeriodos` é uma lista de períodos ($p_{i,i\in\{1,2,3,4\}}$) e `EventosSemSala` é uma lista, ordenada e sem elementos repetidos, de IDs de eventos sem sala nos períodos de `ListaPeriodos`. Deverão ser contabilizados os eventos sem salas associados a disciplinas semestrais (exemplo, p1_2). Isto é verdade para este predicado, mas também para outros predicados em que se peça informação sobre períodos.

Por exemplo,

```
?- eventosSemSalas(Eventos).
Eventos = [14,88,191,311,312,342,343].
?- eventosSemSalasDiaSemana(segunda-feira, Eventos).
Eventos = [191].
?- eventosSemSalasPeriodo([p1], Eventos).
Eventos = [88,191,311,312,342,343].
?- eventosSemSalasPeriodo([], Eventos).
Eventos = [].
```

Em relação ao exemplo anterior, nota que o terceiro pedido é feito sobre o p1, mas o evento 343 é devolvido, pois é um evento sem sala semestral, que ocorre no primeiro semestre (p1_2), pelo que apanha o p1:

```
evento(343,'algebra linear','teorico-pratica',68,semSala).
horario(343,quinta-feira,8.0,10.0,2.0,p1_2).
```

3.2 Pesquisas simples

Recebes um grande agradecimento da Secretaria pelo teu excelente trabalho e preparas-te para voltar ao ~~God of War~~/ver o último episódio do Arcane/~~Rever o Attack on Titan/Outro (risca o que não interessa~~), quando a Área Académica entra em contacto contigo: precisam de ajuda na implementação de um conjunto de predicados. Mais uma vez, sem um suspiro, arregaças as mangas e voltas ao trabalho.

Começas por implementar – **sem usar predicados de ordem superior, ou seja, os predicados que definires têm de usar recursão (tanto faz se geram processos recursivos ou iterativos**) – o predicado `organizaEventos/3`, tal que:

`organizaEventos(ListaEventos, Periodo, EventosNoPeriodo)` é verdade se `EventosNoPeriodo` é a lista, ordenada e sem elementos repetidos, de IDs dos eventos de `ListaEventos` que ocorrem no período `Periodo` para $p_{i,i\in\{1,2,3,4\}}$.

Por exemplo,

```
?- organizaEventos([23, 67, 89, 99, 6], p3, L).
L = [].
```

```
?- organizaEventos([23, 67, 89, 99, 6], p2, L).
L = [6,99].

?- organizaEventos([23, 67, 89, 99, 6], p1, L).
L = [23,67,89,99].
```

Implementas também o predicado `eventosMenoresQue/2`, tal que:

`eventosMenoresQue(Duracao, ListaEventosMenoresQue)` é verdade se `ListaEventosMenoresQue` é a lista ordenada e sem elementos repetidos dos identificadores dos eventos que têm duração menor ou igual a `Duracao`.

Por exemplo:

```
?- eventosMenoresQue(0.5, ListaEventosMenoresQue).
ListaEventosMenoresQue = [4,7].
?- eventosMenoresQue(1.5, ListaEventosMenoresQue).
ListaEventosMenoresQue = [3,4,5,7,...,787,796].
```

De seguida, implementas o predicado `eventosMenoresQueBool/2`, tal que:

`eventosMenoresQueBool(ID, Duracao)` é verdade se o evento identificado por `ID` tiver duração igual ou menor a `Duracao`.

```
?- eventosMenoresQueBool(45, 0.5).
false.
?- eventosMenoresQueBool(4, 0.5).
true.
```

Implementas ainda o predicado `procuraDisciplinas/2`, tal que:

`procuraDisciplinas(Curso, ListaDisciplinas)` é verdade se `ListaDisciplinas` é a lista ordenada alfabeticamente do nome das disciplinas do curso `Curso`.

Por exemplo[2],

```
?- procuraDisciplinas(leti, ListaDisciplinas).
ListaDisciplinas = [algebra linear,
    analise de dados e modelacao estatistica, arquitecturas de redes,
    calculo diferencial e integral i, calculo diferencial e integral iii,
    eletromagnetismo e optica, engenharia de software,
    fundamentos da programacao, gestao,
    introducao a economia,
    introducao a engenharia de telecomunicacoes e informatica,
    introducao aos circuitos e sistemas electronicos,
    mecanica e ondas, programacao com objectos,
    propagacao e antenas, sistemas de comunicacoes,
    sistemas digitais, sistemas operativos].
```

De seguida, implementas – de novo **sem usar predicados de ordem superior, ou seja, os predicados que definires têm de usar recursão (tanto faz se geram processos recursivos ou iterativos**) – o predicado `organizaDisciplinas/3`, tal que:

[2] Por uma questão de legibilidade, por vezes, um predicado é partido em duas linhas. Nota também que a base de conhecimento não está completa, pelo que é normal que faltem disciplinas (exemplo: cálculo II).

`organizaDisciplinas(ListaDisciplinas, Curso, Semestres)` é verdade se `Semestres` é uma lista com duas listas. A lista na primeira posição contém as disciplinas de `ListaDisciplinas` do curso `Curso` que ocorrem no primeiro semestre; idem para a lista na segunda posição, que contém as que ocorrem no segundo semestre. Ambas as listas devem estar ordenadas alfabeticamente e não devem ter elementos repetidos. O predicado falha se não existir no curso `Curso` uma disciplina de `ListaDisciplinas`. Pode-se assumir que não existem disciplinas anuais.

Por exemplo[3],

```
?- organizaDisciplinas([‘algebra linear’,‘compiladores’], ‘leic-t’, L).
L = [[algebra linear],[compiladores]].
?- organizaDisciplinas([‘algebra linear’,‘analitica empresarial’,
   ‘avaliacao de projetos’, ‘ciencia de materiais’], legi, L).
L = [[algebra linear,analitica empresarial,avaliacao de projetos],
    [ciencia de materiais]].
?- organizaDisciplinas([‘algebra linear’,‘analitica empresarial’,
   ‘avaliacao de projetos’, ‘ciencia de materiais’], ’leic-t’, L).
   false.
```

Atacas depois o predicado `horasCurso/5`, tal que:

`horasCurso(Periodo, Curso, Ano, TotalHoras)` é verdade se `TotalHoras` for o número de horas total dos eventos associadas ao curso `Curso`, no ano `Ano` e período `Periodo` = $p_{i,i\in\{1,2,3,4\}}$. Mais uma vez: não esquecer as disciplinas semestrais.

De notar que se vários turnos partilharem o mesmo evento, o número de horas do evento deve contar apenas uma vez. Por exemplo, no caso que se segue devem ser contabilizadas apenas 2 horas:

```
evento(78,’calculo diferencial e integral i’,’teorico-pratica’,86,a1).
horario(78,quarta-feira,8.0,10.0,2.0,p1_2).
turno(78,leti,1,leti0103).
turno(78,leti,1,leti0102).
turno(78,leti,1,leti0101).
```

Por exemplo,

```
?- horasCurso(p1, leic-t’, 1, TotalHoras).
TotalHoras = 50.0.
```

Finalmente, implementas o predicado `evolucaoHorasCurso/2`, tal que:

`evolucaoHorasCurso(Curso, Evolucao)` é verdade se `Evolucao` for uma lista de tuplos na forma (`Ano`, `Periodo`, `NumHoras`), em que `NumHoras` é o total de horas associadas ao curso `Curso`, no ano `Ano` e período `Periodo` ($p_{i,i\in\{1,2,3,4\}}$). `Evolucao` deverá estar ordenada por ano (crescente) e período.

Sugestão: usa o predicado anterior.

Por exemplo,

[3] Nota que leic-t tem de levar plicas. Nota também que se copiar (copy&paste) os objectivos em Prolog a partir do pdf, vai dar um erro de sintaxe no Prolog.

```
?- evolucaoHorasCurso('leic-t', Evolucao).
Evolucao = [(1,p1,50.0),(1,p2,59.0),(1,p3,0),(1,p4,0),
(2,p1,47.0),(2,p2,77.0),(2,p3,0),(2,p4,20.0),
(3,p1,32.0),(3,p2,32.0),(3,p3,39.0),(3,p4,19.0)].
```

3.3 Ocupações críticas de salas

A Área Académica ficou a adorar-te para sempre! Respiras fundo e preparas-te para ir jogar ~~LOL~~/Rocket League/~~COD/Manic Miner/Minecraft/Outro (riscar o que não interessa~~), mas ainda não é desta! A equipa de Gestão do Edifício precisa da tua ajuda: algumas tipologias de salas – por exemplo, anfiteatros – têm uma ocupação intensiva e é preciso identificar quais e quando. Pedem-te para implementar um conjunto de predicados que permita calcular as percentagens de ocupação dos vários tipos de sala, considerando-se ocupações críticas as que ultrapassarem um dado valor (*threshold*). Como já está frio, não arregaças as mangas. Na verdade, não consegues evitar um pequeno suspiro. Mas, logo a seguir, uma onda de energia percorre o teu corpo e lá vais tu, ajudar a equipa de Gestão do Edifício: LET'S GO!!!!!!!!!!!!!

Um evento tem associada uma hora de início e uma hora de fim. Sendo dado um *slot*, com hora de início e hora de fim, este pode ou não cair total ou parcialmente sobre o evento. Assim, implementas o predicado `ocupaSlot/5`, tal que:

`ocupaSlot(HoraInicioDada, HoraFimDada, HoraInicioEvento, HoraFimEvento, Horas)` é verdade se `Horas` for o número de horas sobrepostas (lembrar que 0.5 representa 30 minutos) entre o evento que tem início em `HoraInicioEvento` e fim em `HoraFimEvento`, e o *slot* que tem início em `HoraInicioDada` e fim em `HoraFimDada`. Se não existirem sobreposições o predicado deve falhar (false).

O exemplo que se segue ilustra quatro cenários (no primeiro, o evento fica totalmente contido no *slot*; no segundo, o evento contém totalmente o *slot*; no terceiro, a sobreposição é no início do evento; no quarto, a sobreposição é no fim do evento).

```
?- ocupaSlot(8.5, 11, 9, 10.5, Horas).
Horas = 1.5.
?- ocupaSlot(9.5, 10, 9, 10.5, Horas).
Horas = 0.5.
?- ocupaSlot(8.5, 9.5, 9, 10.5, Horas).
Horas = 0.5.
?- ocupaSlot(10, 11, 9, 10.5, Horas).
Horas = 0.5.
?- ocupaSlot(10, 11, 8, 9, Horas).
false.
```

Implementas de seguida o predicado `numHorasOcupadas/6`[4], tal que:

`numHorasOcupadas(Periodo, TipoSala, DiaSemana, HoraInicio, HoraFim, SomaHoras)` é verdade se `SomaHoras` for o número de horas ocupadas nas salas do tipo `TipoSala`, no intervalo de tempo definido entre `HoraInicio` e `HoraFim`, no dia da semana `DiaSemana`, e no período `Periodo` = $p_{i,i\in\{1,2,3,4\}}$. Não te esqueças das disciplinas semestrais.

Por exemplo,

[4] Os tipos de sala possíveis são os primeiros argumentos do predicado `sala`, no ficheiro keywords.pl (exemplo: grandesAnfiteatros, labsElectro).

```
?- numHorasOcupadas(p1, grandesAnfiteatros, quarta-feira, 8.0, 12.0, S).
S = 6.0.
numHorasOcupadas(p1, grandesAnfiteatros, quarta-feira, 8.0, 10.0, S).
S = 2.5.
```

Sobre o último exemplo, nota que há no a2 (um dos grandes anfiteatros) uma aula de 'Fundamentos da Programação' que apanha apenas 30 minutos do slot 8.0-10.0 (evento 78), ao que se soma as duas horas da aula de 'Cálculo Diferencial e Integral I' (evento 566), no a2.

Implementas ainda o predicado `ocupacaoMax/5`, tal que:

`ocupacaoMax(TipoSala, HoraInicio, HoraFim, Max)` é verdade se `Max` for o número de horas possíveis de ser ocupadas por salas do tipo `TipoSala` (ver acima), no intervalo de tempo definido entre `HoraInicio` e `HoraFim`. Em termos práticos, assume-se que `Max` é o intervalo tempo dado (`HoraFim - HoraInicio`), multiplicado pelo número de salas em jogo do tipo `TipoSala`.

Por exemplo (dado que existem dois grandes anfiteatros),

```
?- ocupacaoMax(grandesAnfiteatros, 8, 12.5, Max).
Max = 9.0.
```

Logo de seguida implementas o predicado `percentagem/3`, tal que:

`percentagem(SomaHoras, Max, Percentagem)` é verdade se `Percentagem` for a divisão de `SomaHoras` por `Max`, multiplicada por 100.

Por exemplo,

```
?- percentagem(5, 9, Percentagem).
Percentagem = 55.55555555555556.
```

Finalmente, implementas o predicado `ocupacaoCritica/4`, tal que:

`ocupacaoCritica(HoraInicio, HoraFim, Threshold, Resultados)` é verdade se `Resultados` for uma lista ordenada de tuplos do tipo `casosCriticos(DiaSemana, TipoSala, Percentagem)` em que `DiaSemana`, `TipoSala` e `Percentagem` são, respectivamente, um dia da semana, um tipo de sala e a sua percentagem de ocupação, no intervalo de tempo entre `HoraInicio` e `HoraFim`, e supondo que a percentagem de ocupação relativa a esses elementos está acima de um dado valor crítico (`Threshold`). Na representação do tuplo, usa o predicado `ceiling` para arredondar para o próximo inteiro o valor da percentagem, isto é `Percentagem` deve ser o primeiro maior inteiro relativo ao valor da percentagem usado nos cálculos (mas apenas na representação do tuplo; nos cálculos deve usar o valor da percentagem sem qualquer arredondamento).

Por exemplo,

```
?- ocupacaoCritica(8, 12.5, 85, Resultados).
Resultados = [casosCriticos(segunda-feira,grandesAnfiteatros,89),
      casosCriticos(segunda-feira,grandesAnfiteatros,95),
      casosCriticos(segunda-feira,pequenosAnfiteatros,93),
      casosCriticos(sexta-feira,labsQuimica,89)].
```

3.4 And now for something completely different...

Depois de teres recebido inúmeros elogios e agradecimentos da Gestão do Edifício chegas a casa. Pensas que é desta que vais ver o "Wednesday/Enola Holmes2/Umbrella Academy/The Boys/~~Altered Carbon~~/Outro (riscar o que não interessa). No entanto, quando estás a abrir a porta da rua, aparece a tua vizinha do lado, Maria de seu nome:

– Jovem, – começa a senhora – já ouvi dizer que é uma divindade da programação e preciso da sua ajuda para organizar a minha família na ceia de Natal. Temos uma mesa de 8 pessoas e seremos 8. Eu e o meu João ocupamos as duas cabeceiras, mas eu tenho de ficar na cabeceira mais próxima da lareira que sou muito friorenta. Vem a Tia Guga, que tem quase 100 anos e tem de ficar à direita do meu João. Depois a minha filha Ana tem de estar ao lado do meu neto Manelito que só tem 3 anos e, do mesmo modo, o meu filho Miguel tem de estar perto do Pedrito. O meu genro Jorge dá-se muito bem com o Miguel e gostaria que ficassem frente a frente na mesa. Acha que tem solução para isto? Ah, esqueci-me de dizer que é muito importante que o Manelito e o Pedrito não fiquem exactamente frente a frente que acabam a atirar batatas e ervilhas um ao outro.

Engoles a seco, lembrando-te que, quando vais de férias, esta senhora fica a tomar conta de Darwin, o teu peixinho laranja, e respondes com o teu melhor sorriso:

– Certo, vou tratar de lhe arranjar um programa que verifique todos esses requisitos.

Entras em casa a pensas que o melhor é implementar algo genérico, não vá a vizinha vir pedir-te soluções sempre que dá um jantar. No entanto, decides assumir (Figura 1) que: a) a mesa de jantar é rectangular, com 8 lugares no total, um lugar em cada cabeceira e 3 em cada lado, estando as cabeceiras da mesa diferenciadas; b) serão exactamente 8 convidados.

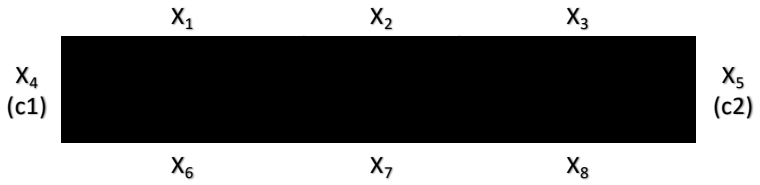

Figura 1: Desenho da mesa

Decides então implementar o predicado `ocupacaoMesa/3`, tal que:

`ocupacaoMesa(ListaPessoas, ListaRestricoes, OcupacaoMesa)` é verdade se `ListaPessoas` for a lista com o nome das pessoas a sentar à mesa, `ListaRestricoes` for a lista de restrições a verificar (ver abaixo) e `OcupacaoMesa` for uma lista com três listas, em que a primeira contém as pessoas de um lado da mesa (X_1, X_2 e X_3), a segunda as pessoas à cabeceira (X_4 e X_5) e a terceira as pessoas do outro lado da mesa (X_6, X_7 e X_8), de modo a que essas pessoas são exactamente as da `ListaPessoas` e verificam todas as restrições de `ListaRestricoes`. Podes assumir que vai haver uma e uma única solução.

Assumes que as restrições possíveis são (exemplos relativos à Figura 1):

- `cab1(NomePessoa)`: é verdade se `NomePessoa` for a pessoa que fica na cabeceira 1 (a que fica perto da lareira) – X_4;
- `cab2(NomePessoa)`: é verdade se `NomePessoa` for a pessoa que fica na cabeceira 2 – X_5;
- `honra(NomePessoa1, NomePessoa2)`: é verdade se `NomePessoa1` estiver numa das cabe-

ceiras e `NomePessoa2` ficar à sua direita – X_3 ou X_6, dependendo da cabeceira ocupada;

- `lado(NomePessoa1, NomePessoa2)`: é verdade se `NomePessoa1` e `NomePessoa2` ficarem lado a lado na mesa[5] – por exemplo, X_7 e X_8;
- `naoLado(NomePessoa1, NomePessoa2)`: é verdade se `NomePessoa1` e `NomePessoa2` não ficarem lado a lado na mesa – por exemplo, X_1 e X_3;
- `frente(NomePessoa1, NomePessoa2)`: é verdade se `NomePessoa1` e `NomePessoa2` ficarem exactamente frente a frente na mesa[6] – por exemplo, X_7 e X_2;
- `naoFrente(NomePessoa1, NomePessoa2)`: é verdade se `NomePessoa1` e `NomePessoa2` não ficarem frente a frente na mesa – por exemplo, X_7 e X_3.

Assim, por exemplo,

```
?- ocupacaoMesa([maria, joao, pedrito, jorge, ana, manelito, miguel, guga],
   [cab1(maria), cab2(joao), honra(joao, guga), lado(ana, manelito),
   lado(miguel, pedrito), frente(miguel, jorge),
   naoFrente(pedrito, manelito)], L).
   L = [[miguel,pedrito,guga],[maria,joao],[jorge,ana,manelito]] ;
   false.
?- ocupacaoMesa([a, b, c, d, e, f, g, h],  [cab1(e), honra(e, b), naoFrente(a, b),
   lado(f, g), lado(a, c), naoLado(f, c), naoFrente(f, c), frente(g, d)], L).
   L = [[c,a,d],[e,h],[b,f,g]] ;
   false.
?- ocupacaoMesa([a, b, c, d, e, f, g, h], [cab1(e), honra(e, b), cab2(c),
   honra(c, a), naoFrente(a, b), naoLado(b, f), lado(f, g), frente(b, h)], L).
   L = [[h,d,a],[e,c],[b,g,f]];
   false.
```

A Figura 2 ilustra a solução para a ceia da vizinha Maria.

Figura 2: Solução para a ceia de Natal da vizinha Maria

Lembras-te também que com o Prolog consegues gerar e testar soluções. Tens é de garantir que não explode com a memória. Pensas que se calhar vais ter de usar functores. Ou se calhar não. Hum...

Quando acabas, vais comunicar a solução à tua vizinha que fica encantada (mais tarde leva-te umas filhoses de agradecimento). Regressas a casa e desligas o telemóvel, *just in case*....

[5] Consideras que quem está nas cabeceiras não está "ao lado" de ninguém.
[6] Consideras que quem está nas cabeceiras não está à frente da pessoa que está na outra cabeceira.

4 Entrega e avaliação

4.1 Condições de realização e prazos

O projecto é realizado individualmente. O código do projecto deve ser entregue obrigatoriamente por via electrónica até às **23h59 de dia 13 de janeiro 2023**, através do sistema Mooshak. Depois desta hora, não serão aceites projectos sob pretexto algum[7]. A ter em conta:

- Deverá ser submetido um ficheiro .pl contendo o código do seu projecto. O ficheiro de código deve conter em comentário, na primeira linha, o número e o nome do aluno;
- Não devem ser utilizados caracteres acentuados ou qualquer caractere que não pertença à tabela ASCII, mesmo em comentários;
- Não esquecer de remover/comentar as mensagens escritas no ecrã;
- A avaliação da execução do código do projecto será feita automaticamente através do sistema Mooshak, usando vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Tipicamente, só poderá efectuar uma nova submissão 15 minutos depois da submissão anterior[8]. Só são permitidas 10 submissões em simultâneo no sistema, pelo que uma submissão poderá ser recusada se este limite for excedido. Nesse caso tente mais tarde;
- Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais. De notar que a base de conhecimento usada nos testes é uma extensão da dada.

Serão publicadas na página da cadeira as instruções necessárias para a submissão do código no Mooshak e a partir dessa altura será possível a submissão por via electrónica. Até ao prazo de entrega poderá efectuar o número de entregas que desejar (por favor, não usar o Mooshak para debug), sendo utilizada para efeitos de avaliação a última entrega efectuada.

Pode ou não haver uma discussão oral do projecto e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

4.2 Cotação

Uma versão estendida do ficheiro dados.pl será usado na avaliação. A nota do projecto será baseada no seguinte:

- Execução correcta – 16 valores distribuídos da seguinte forma:
  1. Qualidade dos dados (2.0 valores)
     (a) `eventosSemSalas`: 0.5 valores;
     (b) `eventosSemSalasDiaSemana`: 0.5 valores;
     (c) `eventosSemSalasPeriodo`: 1.0 valores;
  2. Pesquisa simples (7.5 valores)

[7] Note que o limite de 10 submissões simultâneas no sistema Mooshak implica que, caso haja um número elevado de tentativas de submissão sobre o prazo de entrega, alguns alunos poderão ver-se impossibilitados de submeter o código dentro do prazo.

[8] Note que, se efectuar uma submissão no Mooshak a menos de 15 minutos do prazo de entrega, fica impossibilitado de efectuar qualquer outra submissão posterior.

   (a) `organizaEventos`: 1 valor
   (b) `eventosMenoresQue`: 0.75 valores;
   (c) `eventosMenoresQueBool`: 0.75 valores;
   (d) `procuraDisciplinas`: 1 valor;
   (e) `organizaDisciplinas`: 1.5 valores;
   (f) `horasCurso`: 1.5 valores;
   (g) `evolucaoHorasCurso`: 1 valor;
3. Ocupações críticas das salas (4.5 valores)
   (a) `ocupaSlot`: 1.5 valores;
   (b) `numHorasOcupadas`: 1.0 valores;
   (c) `ocupacaoMax`: 0.75 valores;
   (d) `percentagem`: 0.25 valores;
   (e) `ocupacaoCritica`: 1.0 valor;
4. And now for something completely different... (2 valores)
   (a) `ocupacaoMesa`: 2.0 valores.

- Estilo de programação e facilidade de leitura – 4 valores assim distribuídos:
  - Comentários (1.0 valor): deverão incluir comentários para o utilizador (descrição sumária do predicado); deve também incluir, quando se justifique, comentários para o programador.
  - Boas práticas (3.0 valores):
    * Integração de conhecimento adquirido durante a UC (1.0 valor).
    * Implementação de predicados não excessivamente longos. O facto de usar um predicado recursivo, desde que bem feito, não é penalizado em relação ao uso de predicados funcionais; no entanto, uma má abstração procedimental e duplicações de código serão penalizados (1.0 valor).
    * Escolha de nomes dos predicados auxiliares e das variáveis (1.0 valor).

Presença de *warnings* serão penalizadas (-2 valores). Os predicados 4 e 8, caso não respeitem as indicações dadas na sua implementação, serão penalizados em 1 e 1.5 valores, respectivamente.

4.3 Cópias

Projectos muito semelhantes levarão à reprovação na disciplina e levantamento de processo disciplinar. O corpo docente da disciplina será o único juiz do que se considera copiar.

5 Recomendações

- Recomenda-se o SWI PROLOG, que vai ser usado para a avaliação do projecto.
- Durante o desenvolvimento do programa não se esqueça da Lei de Murphy:
  - Todos os problemas são mais difíceis do que parecem;
  - Tudo demora mais tempo do que pensamos;
  - Se alguma coisa puder correr mal, vai correr mal, na pior das alturas possíveis.