Prova com consulta. Duração: 2h30m. Exame da Época de Recurso

Notas:
a) Responda a cada questão numa folha de exame separada (5 questões = 5 folhas).
b) No Grupo I use apenas Prolog Standard, e no Grupo II use também a biblioteca *CLPFD* do SICStus Prolog.
c) Predicados solicitados em alíneas anteriores podem ser utilizados em alíneas seguintes, mesmo que não os tenha implementado.

GRUPO I – Programação em Prolog

1. [**5 valores**] Considere uma base de conhecimento que indica o dia e a hora a que iniciam as séries de televisão – *airTime(TvShow,DayOfWeek,Hour)* – o número de visualizações (em milhões) que cada série tem – *views(TvShow,MillionsOfViews)* – e o canal em que cada série passa – *network(TvShow,Network)*. Considere ainda que o predicado *hours/1* permite obter uma lista com todas as horas a que podem começar séries nos diferentes canais.

```
hours([7, 7.5, 8, 8.5, 9, 9.5, 10, 10.5]).
```

```
airTime('The Walking Dead',sunday,9).
airTime('Game of Thrones',sunday,8.5).
airTime('The Big Bang Theory',monday,8).
airTime('How I Met Your Mother',thursday,8).
airTime('Mad Men',sunday,10).
```

```
views('The Walking Dead',11).
views('Game of Thrones',5).
views('The Big Bang Theory',9).
views('Mad Men',2.5).
views('How I Met Your Mother',19).
```

```
network('The Walking Dead',amc).
network('Mad Men',amc).
network('Game of Thrones',hbo).
network('The Big Bang Theory',cbs).
network('How I Met Your Mother',cbs).
```

Responda às perguntas seguintes SEM utilizar os predicados *findall*, *setof* e *bagof*.

a) Implemente o predicado ***tvShowNetwork(+Network,+DayOfWeek,+Hour,-TvShow)*** que devolva em *TvShow* qual a série que passa no canal *Network*, no dia *DayOfWeek*, começando à hora *Hour*.

b) Implemente o predicado ***mostViews(+Network,-TvShow,-DayOfWeek,-Hour)*** que devolva em *TvShow* qual a série com mais visualizações do canal *Network* e o seu horário. Em caso de empate qualquer uma serve.

c) Implemente o predicado ***hottestTvShows(+Networks,-TvShows)*** que devolva em *TvShows* a lista com a série com mais visualizações em cada canal indicado na lista *Networks*, pela mesma ordem.

d) Implemente o predicado ***schedule(+Network,+DayOfWeek,-Schedule)*** que devolva em *Schedule* a programação do canal *Network*, no dia *DayOfWeek*, por ordem de transmissão. Nota: não tem que existir programação para todas as horas.

e) Implemente o predicado ***averageViewers(+Network,+DayOfWeek,-Average)*** que devolva em *Average* a média de milhões de visualizações que o canal *Network* tem no dia *DayOfWeek*.

2. [**4 valores**] A empresa MIEIC, Lda. usa um sistema baseado em Prolog para fazer a gestão de tarefas em projetos, representados pelos seguintes tipos de factos:

```
project(ProjID,Name).          % projeto
task(ProjID,TaskId,Description,NecessaryTime).      % tarefa de um projeto
precedence(ProjID,TaskId1,TaskId2).   % precedência entre tarefas (TaskId1->TaskId2)
```

a) Escreva o predicado *proj_tasks(-L)* que obtém uma lista *L* de pares *Project-NTasks*, onde *Project* é o id de um projeto e *NTasks* é o número de tarefas desse projeto. Exemplo:

```
?- proj_tasks(L).
L = [projA-4,projB-3,projC-8]
```

b) Assume-se que todas as tarefas de cada projeto, exceto a inicial, têm pelo menos uma tarefa precedente, e que cada tarefa começa imediatamente após as que a precedem. Cada projeto tem apenas uma tarefa final, caracterizada por não ser precedência de nenhuma outra. Tome o exemplo do projeto *projA*, que tem 4 tarefas, sendo a primeira *t1* e a última *t4*:

```
project(projA,qwe).
task(projA,t1,a,3).          precedence(projA,t1,t2).
task(projA,t2,b,2).          precedence(projA,t1,t3).
task(projA,t3,c,4).          precedence(projA,t2,t4).
task(projA,t4,d,2).          precedence(projA,t3,t4).
```

Examine o seguinte código:

```
p(X,P,D) :- task(X,Y,_,_), \+ precedence(X,_,Y), p(X,Y,P,D).

p(X,Y,[Y|P],D) :- precedence(X,Y,Z), task(X,Y,_,D1),
                  p(X,Z,P,D2), D is D1+D2.
p(X,Y,[Y],D) :- \+ precedence(X,Y,_), task(X,Y,_,D).
```

Diga o que faz o predicado *p/3*. Indique que solução(ões) é(são) encontrada(s) para o objetivo *?- p(projA,P,D)*.

c) Utilizando, se possível, o predicado anterior, escreva o predicado *total_time(+ProjectID,-TotalTime)*, que obtém em *TotalTime* o tempo total necessário para a conclusão do projeto.

3. [**4 valores**] Escreva um programa Prolog que dado um número inteiro positivo *N* e um número de grupos *K*, $K \geq 1$, determina, caso exista, uma distribuição dos números *1,…,N* pelos *K* grupos de tal modo que a soma dos números em cada grupo seja o mesmo. Exemplos:

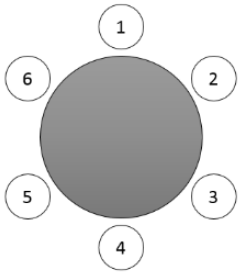
```
?- grupos(5,2,L).
no

?- grupos(5,3,L).
L = [[4,1],[3,2],[5]]

?- grupos(6,2,L).
no

?- grupos(6,3,L).
L = [[6,1],[5,2],[4,3]]

?- grupos(7,2,L).
L = [[7,4,2,1],[6,5,3]]

?- grupos(8,3,L).
L = [[6,3,2,1],[8,4],[7,5]]
```

Prova com consulta. Duração: 2h30m. Exame da Época de Recurso

GRUPO II – Programação em Lógica com Restrições

4. [4 valores]

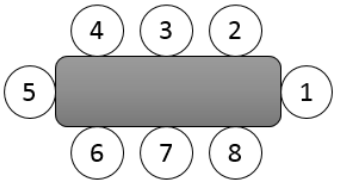
a) Implemente um programa usando PLR que lhe permita obter a posição possível de 6 pessoas numa mesa redonda (representada na figura), obedecendo às seguintes restrições: o Asdrúbal e a Bernardete devem ficar juntos; a Cristalinda e o Demétrio devem ficar juntos; o Eleutério e a Felismina não devem ficar juntos; o Asdrúbal e o Eleutério não devem ficar juntos. Exemplo:




```
?- table6(L).
L = [1, 2, 4, 5, 3, 6] ? ;
L = [1, 2, 5, 4, 3, 6] ? ;
no
```

Note que devem ser descartadas soluções redundantes (mesmas posições relativas, mas rodadas) e simétricas (isto é, L = [1, 6, 4, 3, 5, 2] não deve ser indicada como solução).

b) Generalize o problema da alínea anterior para uma mesa retangular de comprimento par, estendendo as restrições de não adjacência a lugares opostos na mesa (por exemplo, na figura ao lado, os lugares 3 e 7 não deverão ser ocupados por pessoas que não devam ficar juntas). Assuma que o lugar 1 é sempre à cabeceira da mesa, e que os topos da mesa contêm apenas um lugar (não sendo considerados lugares opostos para questões de adjacência). O seu programa deve receber a lista de variáveis que representam as pessoas às quais se quer atribuir um lugar à mesa, o tamanho da mesa, uma lista não vazia de pares de pessoas que devem ficar juntas e uma lista, possivelmente vazia, de pares de pessoas que *não* devem ficar juntas. Exemplo:




```
?- squareTable([A, B, C, D, E, F, G, H], 8,
               [A-B, C-D, E-F, G-H],
               [A-C, B-C, A-E, D-G, D-H, E-G, E-H]).
L = [1, 2, 6, 5, 4, 3, 7, 8]
```

5. [3 valores] Considere que se pretende automatizar o processo de atribuição de serviço docente aos docentes de um departamento. Atente nos seguintes pontos:

- As necessidades de cada disciplina são expressas no predicado *needs(Course,Needs)*, onde *Needs* é o número de horas a alocar. As disciplinas existentes estão no facto *courses([1-plog, 2-esof, 3-laig, 4-ltw, 5-rcom,…])*.
- Um docente pode lecionar entre 6 e 12 horas por semana, tendo idealmente 9 horas atribuídas: a qualidade da solução é prejudicada quando nos desviamos do número ideal. Os docentes existentes estão no facto *teachers([1-eco, 2-aas, 3-jpf, …])*.
- As preferências dos docentes estão expressas no predicado *prefers(Teacher,PrefsList)*, onde o segundo argumento é uma lista de pares disciplina-preferência, sendo a preferência expressa numericamente entre 1 (não quer dar a disciplina) e 5 (quer dar a disciplina). Disciplinas não presentes na lista terão uma classificação por omissão de 3. Uma solução é tanto melhor quanto mais as preferências dos docentes forem satisfeitas.

Esboce o mais detalhadamente possível um programa em PLR que permita obter a distribuição de serviço ótima, atendendo às restrições indicadas.