

[Minha página principal](#) → [Programação em Lógica](#) → [Provas](#) → [Exame da Época Normal](#)

**Data de início** Sexta, 22 Janeiro 2016, 16:37

**Estado** Teste enviado

**Data de submissão:** Sexta, 22 Janeiro 2016, 16:47

**Tempo gasto** 9 minutos 48 segundos

**Nota** 19,90 de um máximo de 20,00 (100%)

Informação

Destacar  
pergunta

No concurso FEUPGotTalent, cada estudante pode participar mostrando as suas habilidades num qualquer tema, académico ou extra-curricular. Os interessados inscrevem-se, dando o número de estudante, idade e o nome da sua atuação:

```
%participant(Id, Age, Performance)
participant(1234, 17, 'Pé coxinho').
participant(3423, 21, 'Programar com os pés').
participant(3788, 20, 'Sing a Bit').
participant(4865, 22, 'Pontes de esparguete').
participant(8937, 19, 'Pontes de pen-drives').
participant(2564, 20, 'Moodle hack').
```

As atuações são apreciadas por um júri de **E** elementos.

Ao longo da atuação (que tem um máximo de 120 segundos), se um elemento do júri achar que o participante não deve passar à próxima fase, carrega num botão. Ficam registados os tempos em que cada elemento do júri carregou no botão. Se não carregou, ficam registados 120 segundos.

```
%performance(Id, Times)
performance(1234, [120, 120, 120, 120]).
performance(3423, [32, 120, 45, 120]).
performance(3788, [110, 2, 6, 43]).
performance(4865, [120, 120, 110, 120]).
performance(8937, [97, 101, 105, 110]).
```


Passam à próxima fase os **N** participantes que mais se aguentaram em palco, somados os tempos de cada elemento do júri, desde que pelo menos um dos elementos do júri não tenha carregado no botão.

Responda às perguntas 1 a 5 **SEM** utilizar predicados de obtenção de múltiplas soluções (findall, setof e bagof).

### Pergunta 1

Respondida

Pontuou 1,00 de 1,00

 Destacar pergunta

Implemente o predicado ***madeItThrough(+Participant)***, que sucede se *Participant* é um participante que já atuou e em cuja atuação pelo menos um elemento do júri não carregou no botão.

```
| ?- madeItThrough(1234).  
yes
```

```
| ?- madeItThrough(2564).  
no
```

```
| ?- madeItThrough(3788).  
no
```


```
madeItThrough(Participant) :-  
    performance(Participant,Times),  
    member(120,Times).
```

Comentário:

### Pergunta 2

Respondida

Pontuou 1,50 de 1,50

 Destacar pergunta

Implemente o predicado ***juriTimes(+Participants, +JuriMember, -Times, -Total)***, que devolve em *Times* o tempo de atuação de cada participante na lista *Participants* (pela mesma ordem) até que o júri número *JuriMember* (de 1 a E) carregou no botão, e em *Total* a soma desses tempos.

```
| ?- juriTimes([1234,3423,3788,4865,8937],1,Times,Total).  
Times = [120,32,110,120,97],  
Total = 479
```

```
| ?- juriTimes([1234,3423,3788,4865,8937],2,Times,Total).  
Times = [120,120,2,120,101],  
Total = 463
```

```
:- use_module(library(lists)).
```


```
juriTimes([Participant|Participants], JuriMember, [Time|Times], Total) :-  
    performance(Participant,PTimes),  
    nth1(JuriMember, PTimes, Time),  
    juriTimes(Participants,JuriMember,Times,Total2),  
    Total is Total2 + Time.  
juriTimes([],_,[],0).
```

Comentário:

Pergunta **3**

Respondida

Pontuou 1,00 de 1,00

 Destacar pergunta

Implemente o predicado **patientJuri(+JuriMember)** que sucede se o júri *JuriMember* já se absteve de carregar no botão pelo menos por duas vezes.

```
| ?- patientJuri(3).  
no
```

```
| ?- patientJuri(4).  
yes
```

```
:- use_module(library(lists)).
```


```
patientJuri(JuriMember) :-  
    performance(P1,Times1),  
    performance(P2,Times2),  
    P1 \= P2,  
    nth1(JuriMember, Times1, 120),  
    nth1(JuriMember, Times2, 120).
```

Comentário:

Pergunta **4**

Respondida

Pontuou 1,50 de 1,50

 Destacar pergunta

Implemente o predicado **bestParticipant(+P1, +P2, -P)** que unifica *P* com o melhor dos dois participantes *P1* e *P2*. O melhor participante é aquele que tem uma maior soma de tempos na sua atuação (independentemente de estar ou não em condições de passar à próxima fase). Se ambos tiverem o mesmo tempo total, o predicado deve falhar.

```
| ?- bestParticipant(3423,1234,Z).  
Z = 1234
```

```
| ?- bestParticipant(1234,1234,Z).  
no
```

```
:- use_module(library(lists)).
```

```
bestParticipant(P1, P2, P) :-  
    performance(P1,Times1),  
    performance(P2,Times2),  
    sumlist(Times1,TotalTime1),
```

```
sumlist(Times2,TotalTime2),  
bestParticipant(P1, TotalTime1, P2, TotalTime2, P).
```

```
bestParticipant(P1, TotalTime1, _, TotalTime2, P1) :-  
    TotalTime1 > TotalTime2.
```


```
bestParticipant(_, TotalTime1, P2, TotalTime2, P2) :-  
    TotalTime2 > TotalTime1.
```

Comentário:

Pergunta **5**

Respondida

Pontuou 1,00 de  
1,00

 Destacar  
pergunta


Implemente o predicado ***allPerfs***, que imprime na consola os números dos participantes que já atuaram, juntamente com o nome da sua atuação e lista de tempos.

```
| ?- allPerfs.  
1234:Pé coxinho:[120,120,120,120]  
3423:Programar com os pés:[32,120,45,120]  
3788:Sing a Bit:[110,2,6,43]  
4865:Pontes de esparguete:[120,120,110,120]  
8937:Pontes de pen-drives:[97,101,105,110]  
yes
```

```
allPerfs :-  
    participant(Id,_,Performance),  
    performance(Id,Times),  
    format("~w:~w:~w~N",[Id,Performance,Times]),  
    fail.  
allPerfs.
```

Comentário:

Informação


 Destacar  
pergunta

Nas perguntas seguintes pode fazer uso de predicados de obtenção de múltiplas soluções (findall, setof e bagof).

Pergunta **6**

Respondida

Pontuou 1,00 de  
1,00

 Destacar  
pergunta

Implemente o predicado ***nSuccessfulParticipants(-T)*** que determina quantos participantes não tiveram qualquer clique no botão durante a sua atuação.

```
| ?- nSuccessfulParticipants(T).  
T = 1
```

```
successfulParticipant(Participant) :-  
    performance(Participant,Times),  
    \+ (member(Time,Times),  
        Time \= 120).
```


```
nSuccessfulParticipants(T) :-  
    findall(Participant,successfulParticipant(Participant),List),  
    length(List,T).
```

Comentário:

Pergunta **7**

Respondida

Pontuou 1,40 de  
1,50

 Destacar  
pergunta

Implemente o predicado **juriFans(juriFansList)**, que obtém uma lista contendo, para cada participante, a lista dos elementos do júri que não carregaram no botão ao longo da sua atuação.

```
| ?- juriFans(L).  
L = [1234-[1,2,3,4],3423-[2,4],3788-[],4865-[1,2,4],8937-[]]
```

```
participantJuriFans([120 | Times],Jurild,[Jurild | JuriFans]) :-  
    !,  
    Jurild1 is Jurild + 1,  
    participantJuriFans(Times,Jurild1,JuriFans).  
participantJuriFans([_ | Times],Jurild,JuriFans) :-  
    Jurild1 is Jurild + 1,  
    participantJuriFans(Times,Jurild1,JuriFans).  
participantJuriFans([],_,[]).
```

```
juriFansAux([Participant | Participants],[Participant-Fans | JuriFansList]) :-  
    performance(Participant,Times),  
    participantJuriFans(Times,1,Fans),  
    juriFansAux(Participants,JuriFansList).  
juriFansAux([],[]).
```

```
juriFans(JuriFansList) :-  
    findall(Participant,performance(Participant, _),Participants),  
    juriFansAux(Participants,JuriFansList).
```

Comentário:  
podia tirar mais partido de findall...

Pergunta **8**

Respondida

Pontuou 1,50 de  
1,50

O seguinte predicado permite obter participantes, suas atuações e tempos totais, que estejam em condições de passar à próxima fase: para um participante poder passar, tem de haver pelo menos um

elemento do júri que não tenha carregado no botão durante a sua atuação.

```
:- use_module(library(lists)).
```

```
eligibleOutcome(Id,Perf,TT) :-  
    performance(Id,Times),  
    madeItThrough(Id),  
    participant(Id,_,Perf),  
    sumlist(Times,TT).
```

Fazendo uso deste predicado, implemente o predicado **nextPhase(+N, -Participants)**, que obtém a lista com os tempos totais, números e atuações dos *N* melhores participantes, que passarão portanto à próxima fase. Se não houver pelo menos *N* participantes a passar, o predicado deve falhar.

```
| ?- nextPhase(2,P).  
P = [480-1234-'Pé coxinho',470-4865-'Pontes de esparguete']
```

```
| ?- nextPhase(3,P).  
P = [480-1234-'Pé coxinho',470-4865-'Pontes de esparguete',3  
17-3423-'Programar com os pés']
```

```
| ?- nextPhase(4,P).  
no
```

```
:- use_module(library(lists)).
```


```
nextPhase(N, Participants) :-  
    setof(TT-Id-Perf,eligibleOutcome(Id,Perf,TT),Results),  
    reverse(Results,Ordered),  
    sublist(Ordered,Participants,0,N,_).
```

Comentário:

Pergunta **9**

Respondida

Pontuou 1,00 de  
1,00

 Destacar  
pergunta

Explique o que faz o predicado **predX/3** apresentado abaixo. Indique ainda se o *cut* utilizado é verde ou vermelho, justificando a sua resposta.

```
predX(Q,[R|Rs],[P|Ps]) :-  
    participant(R,I,P), I=<Q, !,  
    predX(Q,Rs,Ps).  
predX(Q,[R|Rs],Ps) :-  
    participant(R,I,_), I>Q,  
    predX(Q,Rs,Ps).  
predX(_,[],[]).
```

Para uma dada lista de participantes, dá a lista de performances dos participantes com idade menor ou igual à idade escolhida.


O cut é verde visto que não altera a lógica do programa, apenas reduz backtracking desnecessário.

Isto deve-se ao facto de no segundo predicado existir a verificação " $I>Q$ ", a inversa da do predicado acima, " $I\leq Q$ ".

O cut não seria verde se não existisse a verificação " $I>Q$ ", pois nesse caso o backtracking geraria respostas erradas.

Comentário:

Informação


 Destacar  
pergunta

Dado um número **N**, pretende-se determinar uma sequência de  $2*N$  números que contenha, para todo o  $k \in [1,N]$ , uma sub-sequência  $S_k = k, \dots, k$  começada e terminada com o número  $k$  e com  $k$  outros números de permeio. Por exemplo, a sequência  $[2, 3, 1, 2, 1, 3]$  cumpre os requisitos: os 1s têm 1 número no meio, os 2s têm 2 números no meio, e os 3s têm 3 números no meio. A sequência  $[2, 3, 4, 2, 1, 3, 1, 4]$  também cumpre. No entanto, alguns valores de  $N$  não têm solução possível.

Pergunta **10**

Respondida

Pontuou 1,00 de  
1,00

 Destacar  
pergunta

Explique o que faz o seguinte predicado:

```
impoe(X,L) :-  
    length(Mid,X),  
    append(L1,[X|_],L), append(_,[_|Mid],L1).
```

Este predicado faz com que na lista  $L$  existam dois número  $N$ , separados por  $N$  algarismos.

Através de backtracking, gera todas as hipóteses possíveis para esta condição.


Comentário:

$N=X$

## Pergunta 11

Respondida

Pontuou 1,50 de 1,50

 Destacar pergunta

Tirando partido do predicado anterior, implemente o predicado **langford(+N,-L)**, em que **N** é um inteiro dado e **L** será uma sequência de  $2*N$  números conforme indicado atrás. (Nota: Langford foi o matemático escocês que propôs este problema.)

```
| ?- langford(3,L).
L = [3,1,2,1,3,2] ? ;
L = [2,3,1,2,1,3] ? ;
no
```

```
| ?- langford(4,L).
L = [4,1,3,1,2,4,3,2] ? ;
L = [2,3,4,2,1,3,1,4] ? ;
no
```


```
| ?- langford(5,L).
no
```

```
impoe_n(X,L,N) :-
    X <= N,
    impoe(X,L),
    X1 is X + 1,
    impoe_n(X1,L,N).
impoe_n(X,_N) :-
    X > N.
```

```
langford(N,L) :-
    Length is 2*N,
    length(L,Length),
    impoe_n(1,L,N).
```

Comentário:

Informação


 Destacar pergunta

No seguinte problema de Programação em Lógica com Restrições deve usar a biblioteca **clpfd** do SICStus Prolog.

## Pergunta 12

Respondida

Pontuou 3,50 de 3,50

 Destacar pergunta

Construa em PLR o programa **ups\_and\_downs(+Min,+Max,+N,-L)**, que gera uma sequência de números oscilantes, isto é, na qual não haja sequências de três valores ordenados (crescente ou decrescentemente). Ou seja, cada valor deverá ser menor do que os dois valores adjacentes ou maior do que os dois valores adjacentes. Os valores estão compreendidos entre **Min** e **Max** (ambos positivos). A lista **L** com os valores gerados deve ter comprimento **N >= 1**.



```
| ?- ups_and_downs(1,2,2,L).
```

```
L = [1,2] ? ;
```

```
L = [2,1] ? ;
```

```
no
```

```
| ?- ups_and_downs(1,2,3,L).
```

```
L = [1,2,1] ? ;
```

```
L = [2,1,2] ? ;
```

```
no
```

```
| ?- L = [_X1,_X2,_,_], _X1#<_X2, all_different(L), ups_and_downs(1,4,4,L).
```

```
L = [1,3,2,4] ? ;
```

```
L = [1,4,2,3] ? ;
```

```
L = [2,3,1,4] ? ;
```

```
L = [2,4,1,3] ? ;
```

```
L = [3,4,1,2] ? ;
```

```
no
```

```
| ?- L = [3,_,_,_], all_different(L), ups_and_downs(1,4,4,L).
```

```
L = [3,1,4,2] ? ;
```

```
L = [3,2,4,1] ? ;
```

```
L = [3,4,1,2] ? ;
```

```
no
```

```
| ?- length(L,5), circuit(L), ups_and_downs(1,5,5,L).
```

```
L = [2,4,1,5,3] ? ;
```

```
L = [3,1,5,2,4] ? ;
```

```
L = [3,4,2,5,1] ? ;
```

```
L = [4,5,2,3,1] ? ;
```

```
L = [5,1,4,2,3] ? ;
```

```
L = [5,3,4,1,2] ? ;
```

```
no
```

```
:- use_module(library(clpfd)).
```

```
ups_and_downs(Min,Max,N,L) :-
```

```
length(L,N),
```

```
domain(L,Min,Max),
```

```
restricoes(L),
```

```
!,
```

```
labeling([],L).
```

```
restricoes([_]).
```

```
restricoes([A,B,C | Lista]) :-
```


```
(B #< A #/\ B #< C) #V (B #> A #/\ B #> C),
```

```
restricoes([B,C | Lista]).
```

restricoes([A,B]) :-  
B #< A #\ V B #> A.

Comentário:

Informação


 Destacar  
pergunta

No seguinte problema de Otimização deve usar a biblioteca **clpfd** do SICStus Prolog.

Pergunta **13**

Respondida

Pontuou 3,00 de  
3,00

 Destacar  
pergunta

Nas eleições presidenciais de um qualquer país à beira mar plantado, um candidato vê-se a braços com a decisão de escolher quais concelhos do país percorrer, com o objetivo de angariar o maior número possível de votos. Como a esposa deste candidato é muito desconfiada, exige sempre que após um comício num desses concelhos regresse a casa para jantar. Cada concelho está representado por factos do tipo **concelho(Nome,Distancia,NEleitoresIndecisos)**, onde a distância já contempla ida e volta:

```
concelho(x,120,410).  
concelho(y,10,800).  
concelho(z,543,2387).  
concelho(w,3,38).  
concelho(k,234,376).
```

Dado um número de dias de campanha, o candidato quer saber quais concelhos deve visitar (no máximo 1 por dia) de modo a maximizar o número de eleitores indecisos nos concelhos que visite. Paralelamente, dado o seu estado de idade avançado, pretende aplicar um limite à distância total percorrida em toda a campanha: qualquer solução válida não poderá ultrapassar esse limite.

Construa um programa em PLR que ajude o candidato a tomar a melhor decisão. Para tal, defina o predicado **concelhos(+NDias,+MaxDist,-ConcelhosVisitados,-DistTotal,-TotalEleitores)**: dado o número de dias *NDias* da campanha e a distância máxima *MaxDist* a percorrer, devolve em *ConcelhosVisitados* a lista dos concelhos a visitar, em *DistTotal* a distância total a percorrer e em *TotalEleitores* o número total de eleitores indecisos abrangidos por tais visitas.

```
| ?- concelhos(3,700,CVs,Dist,TE).  
TE = 3597,  
CVs = [x,y,z],  
Dist = 673
```

```
| ?- concelhos(3,500,CVs,Dist,TE).
```

```
TE = 1586,
```

```
CVs = [x,y,k],
```

```
Dist = 364
```

```
| ?- concelhos(4,500,CVs,Dist,TE).
```

```
TE = 1624,
```

```
CVs = [x,y,w,k],
```

```
Dist = 367
```

```
| ?- concelhos(4,290,CVs,Dist,TE).
```

```
TE = 1248,
```

```
CVs = [x,y,w],
```

```
Dist = 133
```

```
:- use_module(library(clpfd)).
```

```
concelhos(NDias,MaxDist,ConcelhosVisitados,DistTotal,TotalEleitores) :-
```

```
    findall(Concelho-Distancia-
```

```
    NEleitoresIndecisos,concelho(Concelho,Distancia,NEleitoresIndecisos),Concelhos),
```

```
    length(Concelhos,Nconcelhos),
```

```
    length(Visitas,Nconcelhos),
```

```
    domain(Visitas,0,1),
```

```
    count(1,Visitas,#=,ContagemDias),
```

```
    ContagemDias #=< NDias,
```

```
    contagemEleitores(Visitas,Concelhos,TotalEleitores),
```

```
    distanciaTotal(Visitas,Concelhos,DistTotal),
```

```
    DistTotal #=< MaxDist,
```

```
    labeling([maximize(TotalEleitores)],Visitas),
```

```
    gerarConcelhosVisitados(Concelhos,Visitas,ConcelhosVisitados).
```

```
gerarConcelhosVisitados([Concelho-_- | Concelhos],[1 | Visitas],
```

```
[Concelho | ConcelhosVisitados]) :-
```

```
    gerarConcelhosVisitados(Concelhos,Visitas,ConcelhosVisitados).
```

```
gerarConcelhosVisitados(_ | Concelhos],[0 | Visitas],ConcelhosVisitados) :-
```

```
    gerarConcelhosVisitados(Concelhos,Visitas,ConcelhosVisitados).
```

```
gerarConcelhosVisitados([],[],[]).
```

```
contagemEleitores([Visitado | Visitas],[_ -
```

```
Nindecisos | ListaIndecicos],Total) :-
```

```
    contagemEleitores(Visitas,ListaIndecicos,Total2),
```

```
    Total #= Total2 + Visitado*Nindecisos.
```

```
contagemEleitores([],[],0).
```

```
distanciaTotal([Visitado | Visitas],[_ -Distancia-_| Distancias],DistTotal) :-
```

```
    distanciaTotal(Visitas,Distancias,DistTotal2),
```

```
    DistTotal #= DistTotal2 + Visitado*Distancia.
```

```
distanciaTotal([],[],0).
```