voo(IDvoo, O, D, Partida, Chegada).

```
voo(fr5483,porto,madrid,500,620).
voo(fr131,porto,colonia,505,690).
voo(fr8862,porto,malaga,505,635).
voo(fr9135,porto,paris,550,720).
voo(fr8348,porto,londres,1085,1210).
%...
voo(fr1225,madrid,colonia,725,860).
voo(fr5995,madrid,londres,785,870).
voo(fr5493,madrid,milao,810,950).
%...
voo(fr2508,colonia,barcelona,885,995).
voo(fr8518,colonia,milao,940,1005).
voo(fr2817,colonia,londres,1015,1015).
%...
voo(fr4195,milao,londres,1070,1120).
```

a)[30%] Escreva um predicado *encontra_voos(Origem,Destino,T,L)* , onde *T* é o instante de tempo a partir do qual o passageiro consegue aceder à zona de embarque/desembarque do aeroporto (também em minutos a partir das 0:00). Admita

que qualquer ligação num dado aeroporto exige pelo menos 60 minutos entre a chegada à zona de embarque/desembarque e a partida dos voos correspondentes. Os elementos de *L* deverão conter a identificação, *IDvoo*, dos voos usados.

```
encontra_voos(Origem,Destino,T,L):-encontra_voos2(Origem,Destino,[Origem], T, L).

encontra_voos2(Destino,Destino,_, _,[]):-!.
encontra_voos2(Origem,Destino,L, T,[ID|RPercurso]):-
                                 voo(ID,Origem,X,TP,TC),
                                 Taux is T+60, TP >= Taux,
                                 not(member(X,L)),
                                 encontra_voos2(X,Destino,[X|L], TC, RPercurso).
```

b)[10%] Escreva um predicado *todas_alternativas_voo(O,D,T,LL)* que encontre todas as alternativas de voo entre a origem *O* e o destino *D* a partir do instante *T* gerando a lista *LL*, em que as listas internas são possibilidades de voos usadas (idênticas à lista *L* da alínea anterior).

```
todas_alternativas_voo(O,D,T,LL):-
        findall(L,encontra_voos(O,D,T,L),LL).
```

c)[30%] Escreva um predicado *chega_mais_cedo(O,D,T,L)*, que gera uma lista *L* correspondente aos voos usados que permitem chegar mais cedo ao destino. *O*, *D* e *T* têm o mesmo significado da alínea anterior.

```
chega_mais_cedo(O,D,T,F):-
        todas_alternativas_voo(O,D,T,[X|L]),
        mais_cedo(L,X,F).


mais_cedo([],X,X).

mais_cedo([Y|R],X,Z):-
        ultimo_elemento(Y,Y1),
        ultimo_elemento(X,X1),
        voo(X1,_,_,_,T1),
        voo(Y1,_,_,_,T2),
        T1<T2,!,
        mais_cedo(R,X,Z).

mais_cedo([Y|R],_,Z):-
        mais_cedo(R,Y,Z).

ultimo_elemento([X],X).
ultimo_elemento([_|L],X):-
        ultimo_elemento(L,X).
```

d)[30%] Escreva um predicado *menos_ligacoes(O,D,T,L)*, que gera uma lista *L* que corresponde à solução ou uma das soluções onde haja menos ligações entre voos.

```
menos_ligacoes(O,D,T,F):-
        todas_alternativas_voo(O,D,T,[X|L]),
        menos_lig(L,X,F).

menos_lig([],X,X).

menos_lig([Y|R],X,Z):-
        length(X,X1),
        length(Y,Y1),
        X1 < Y1,!,
        menos_lig(R,X,Z).

menos_lig([Y|R],_,Z):-
        menos_lig(R,Y,Z).
```

liga(divisão1,divisão2,largura,altura),

```
liga(hall,corredor,90,190).
liga(hall,sala,160,190).
liga(sala,cozinha,90,190).
liga(cozinha,corredor,100,200).
liga(cozinha,lavandaria,80,180).
liga(cozinha,arrumos,80,180).
liga(corredor,wc,90,200).
liga(corredor,quarto1,95,200).
liga(corredor,quarto2,95,200).
liga(corredor,quarto_suite,120,200).
liga(quarto_suite,wc_suite,90,200).
liga(quarto_suite,terraço,200,200).
liga(sala,terraço,240,200).
```

a)[15%] Escreva um predicado *ppp(X,Y,Z,LP,AP,T)* que verifique se uma dada caixa com dimensões (*X,Y,Z*), cujas arestas com dimensão *Z* são perpendiculares ao chão, passa por uma porta com largura *LP* e altura *AP*. Note que a caixa pode ser tombada. *T* deverá retornar com valor *n* caso não seja necessário tombar a caixa, com *x* caso seja necessário tombá-la de modo a pôr as arestas com dimensão *X* na perpendicular ao chão ou com *y* caso seja necessário pôr as arestas com dimensão *Y* na perpendicular ao chão.

```
ppp(X,Y,Z,LP,AP,n):- Z=<AP, (X=<LP; Y=<LP),!.
ppp(X,Y,Z,LP,AP,x):- X=<AP, (Y=<LP; Z=<LP),!.
ppp(X,Y,Z,LP,AP,y):- Y=<AP, (X=<LP; Z=<LP).
```

b)[35%] Escreva *caminho(X,Y,Z,Div_Partida,Div_Chegada,Lista_Divisões,N)*, um predicado que forneça o caminho para levar uma caixa com dimensões (*X,Y,Z*) da divisão *Div_Partida* até a divisão *Div_Chegada*, retornando em *Lista_Divisões* todas as divisões por onde passou (incluindo a de partida e a de chegada). Note que o caminho deve permitir que a caixa passe pelas portas. Em *N* fica o número de vezes que foi necessário tombar a caixa.

```
caminho(X,Y,Z,Div_Partida,Div_Chegada,Lista_Divisoes,N):-
        caminho2(X,Y,Z,Div_Chegada,[Div_Partida],Lista_Divisoes,0,N).

caminho2(_,_,_,Dest,[Dest|T],[Dest|T],N,N).
caminho2(X,Y,Z,Dest,[H|T],LD,NT,N):-
        %evitar backtracking
        H\==Dest,
        %ligação bi-direccional
        (liga(H,NDiv,LP,AP);liga(NDiv,H,LP,AP)),
        %evitar caminhos circulares
        \+ member(NDiv,[H|T]),
        %a porta permite passagem
        %testar tipo de tombo para actualizar coordenadas e contar tombos
        ppp(X,Y,Z,LP,AP,Tombo),
        (
                (Tombo==n,
                 caminho2(X,Y,Z,Dest,[NDiv,H|T],LD,NT,N));
                (Tombo==x,
                 NT1 is NT+1,
                 caminho2(Z,Y,X,Dest,[NDiv,H|T],LD,NT1,N));
                (Tombo==y,
                 NT1 is NT+1,
                 caminho2(X,Z,Y,Dest,[NDiv,H|T],LD,NT1,N))
        ).
```

c)[10%] Considerando que o predicado caminho/7 já foi implementado, implemente agora o predicado *todos_caminhos(X,Y,Z,Div_Partida,Div_Chegada,LLista_Divisões)* em que *LLista_Divisões* é uma lista de pares *p(N,Lista_Divisões)*, onde *N* e *Lista_Divisões* têm o significado dado na alínea anterior.

```
todos_caminhos(X,Y,Z,Div_Partida,Div_Chegada,LLista_Divisoes):-
   findall( p(N,Lista_Divisoes),
               caminho(X,Y,Z,Div_Partida,Div_Chegada,Lista_Divisoes,N),
               LLista_Divisoes).
```

d)[40%] Implemente *menosdivtomb(X,Y,Z,Div_Partida,Div_Chegada,Lista_Divisões)*, um predicado que determina o caminho, ou um dos caminhos, entre *Div_Partida* e *Div_Chegada* que passe pelo menor número de divisões. Em caso de igualdade deverá ser dada prioridade à solução, ou a uma das soluções, que envolva tombar menos vezes a caixa.

```
menosdivtomb(X,Y,Z,Div_Partida,Div_Chegada,Lista_Divisoes):-
findall( (ND,NT,LD),
(caminho(X,Y,Z,Div_Partida,Div_Chegada,LD,NT),length(LD,ND)),
LLista_Divisoes),
%em caso de igualdade o segundo campo (n?mero de tombos) serve de desempate
sort(LLista_Divisoes,[(_,_,Lista_Divisoes)|_]).
```

```
compra(1,[leite,cha,bolo]).
compra(2,[ovos,cha,refrigerante]).
compra(3,[leite,ovos,cha,refrigerante]).
compra(4,[ovos,refrigerante]).
compra(5,[sumo]).
```

a)[15%] Escreva um predicado *todos_produtos(LTP)* que coloque na lista *LTP* todos os produtos indicados nas compras. Um dado produto só deve aparecer uma vez nessa lista. Para o exemplo considerado teríamos *LTP = [bolo, leite, cha, ovos, refrigerante, sumo]*.

```
todos_produtos(LTP):-
        findall(P,(compra(_,LPC),member(P,LPC)),LTPR),  elimina_repetidos(LTPR,LTP).

elimina_repetidos([],[]).
elimina_repetidos([X|L],L1):-member(X,L),!,elimina_repetidos(L,L1).
elimina_repetidos([X|L],[X|L1]):-elimina_repetidos(L,L1).
```

b)[15%] Escreva um predicado *nivel_suporte(LTPNS)* que retorna uma lista *LTPNS* de termos do tipo *p(P,Qt)* que representam a quantidade *Qt* de compras (nível de suporte) envolvendo o produto *P*. Para o exemplo considerado teríamos *LTPNS = [p(bolo, 1), p(leite, 2), p(cha, 3), p(ovos, 3), p(refrigerante, 3), p(sumo, 1)]*.

```
nivel_suporte(LTPNS):-                  todos_produtos(LTP),nivel_suporte1(LTP,LTPNS).

nivel_suporte1([],[]).
nivel_suporte1([P|LP],[p(P,QtI)|LTPNS]):-findall(I,(compra(I,LPI),member(P,LPI)),LI), length(LI,QtI),
                            nivel_suporte1(LP,LTPNS).
```

c)[10%] Escreva um predicado *cortainferioresN1(LTPNS,N1,LP)* que retorna uma lista *LP* com os nomes dos produtos que têm pelo menos o nível de suporte *N1*. *LTPNS* é uma lista idêntica à obtida no predicado da alínea anterior. Para o exemplo considerado, admitindo *N1* igual a 2, teríamos *LP = [leite, cha, ovos, refrigerante]*.

```
cortainferioresN1([],_,[]).
cortainferioresN1([p(_,Qt)|LTPNS],N1,LP):-      Qt<N1,!,cortainferioresN1(LTPNS,N1,LP).
cortainferioresN1([p(P,_)|LTPNS],N1,[P|LP]):-   cortainferioresN1(LTPNS,N1,LP).
```

d)[10%] Escreva um predicado *escolheN(N,LP,LPC)* que a partir de uma lista de produtos *LP* idêntica à obtida na alínea anterior, retorna uma lista *LPC* com uma combinação de *N* produtos dessa lista. Se chamarmos esse predicado com *N* igual a 3 e com a lista *LP* obtida na alínea anterior teríamos como primeira solução *LPC = [leite, cha, ovos]* e se pedíssemos novas soluções com o ";" obteríamos ainda: *LPC = [leite, cha, refrigerante]* ; *LPC = [leite, ovos, refrigerante]* ; *LPC = [cha, ovos, refrigerante]*.

```
escolheN(0,_,[]):-!.
escolheN(N,[X|L],[X|L1]):- N1 is N-1,escolheN(N1,L,L1).
escolheN(N,[_|L],L1):- escolheN(N,L,L1).
```

e)[50%] Escreva um predicado *selecionaNsuporteN1(N,N1,Lf)* que retorna uma lista *Lf* com listas de *N* produtos que tenham o suporte de pelo menos *N1*. Seguem-se os resultados de 3 chamadas deste predicado:

?- selecionaNsuporteN1(3,2,Lf).
Lf = [[cha, ovos, refrigerante]].

?- selecionaNsuporteN1(2,3,Lf).
Lf = [[ovos, refrigerante]].

?- selecionaNsuporteN1(2,2,Lf).
Lf = [[leite, cha], [cha, ovos], [cha, refrigerante], [ovos, refrigerante]].

```
selecionaNsuporteN1(N,N1,Lf):-
                        nivel_suporte(LTPNS),
                        cortainferioresN1(LTPNS,N1,LP),
                        findall(LPC,escolheN(N,LP,LPC),LLPC),
                        findall(LC,compra(_,LC),LLC),
                        verifica(LLPC,LLC,N1,Lf).


verifica([],_,_,[]).
verifica([LPC|LLPC],LLC,N1,[LPC|Lf]):-                          pertenceN2(LPC,LLC,N2),N2>=N1,!,
        verifica(LLPC,LLC,N1,Lf).
verifica([_|LLPC],LLC,N1,Lf):-verifica(LLPC,LLC,N1,Lf).

pertenceN2(_,[],0).
pertenceN2(LPC,[LC|LLC],N2):-intersecao(LPC,LC,LPC),!,
                             pertenceN2(LPC,LLC,N),N2 is N+1.
pertenceN2(LPC,[_|LLC],N2):-pertenceN2(LPC,LLC,N2).


intersecao([], _,[]).
intersecao([X|L1], L2, [X|LR]) :- member(X,L2), !,                                  intersecao(L1,L2,LR).
intersecao([_|L1], L2, LR) :-intersecao(L1,L2,LR).
```