BE4 - Programmation Logique avec Contraintes

Algorithme et raisonnement

Emeline GOT - Théo PONTON

Table des matières

1 Problème de localisation d'entrepôts

On souhaite servir 5 clients différents à moindre coût à partir de 3 entrepôts. Pour cela, on considère que l'implantation des entrepôts produit des coûts d'ouverture et de maintenance (coût fixe) et que chaque livraison d'un entrepôt à un client produit des coûts de transport.

On suit les étapes du schéma CSP pour résoudre le problème.

Dans un premier temps, on définit les variables. La variable C regroupe tous les Cij qui sont associés à un client i et à un entrepôt j. Les Cij valent 1 si le client i est fourni par l'entrepôt j, et 0 sinon. La variable E regroupe les 3 coûts fixes associés aux 3 entrepôts. La variable M donne la valeur des coûts de transports de chaque entrepôt à chaque client.

```
C=[C11,C12,C13,
C21,C22,C23,
C31,C32,C33,
C41,C42,C43,
C51,C52,C53],

E=[18,10,28],

M=[5,4,0,0,3,
7,0,2,0,0,
0,5,5,4,8]
```

Ensuite, on définit les domaines des variables. Seul le domaine des éléments de la liste C doit être définit. Les Cij étant compris entre 0 et 1, on indique :

```
C::0..1
```

On définit ensuite les relations entre les variables. Les contraintes à respecter sont les suivantes :

— Un client n'est servi que par un seul entrepôt

```
C11+C12+C13#=1,
C21+C22+C23#=1,
C31+C32+C33#=1,
C41+C42+C43#=1,
C51+C52+C53#=1
```

— Un entrepôt non sélectionné ne doit pas figurer comme fournisseur d'un client

```
F1#=C11+C21+C31+C41+C51,
F2#=C12+C22+C32+C42+C52,
F3#=C13+C23+C33+C43+C53
```

On définit la liste F correspondant à la sélection d'un entrepôt ou non. Si Fij ne vaut pas 0, alors l'entrepôt est utilisé pour servir les clients.

```
F=[F1,F2,F3]
```

Pour la résolution du problème, on s'intéresse donc à minimiser les coûts de transports et les coûts fixes. Pour cela, on utilise les prédicats *calcul_cout_transport* et *calcul_cout_fixe* qui calculent les valeurs de ces deux coûts.

— ***calcul_cout_transport*** :

```
calcul_cout_transport([],[],0). %Cas d'arrêt
calcul_cout_transport([0|R],[T|Q],C):- calcul_cout_transport(R,Q,C).
    %Si le lien client - entrepôt ij n'est pas utilisé (Cij = 0), on
    continue le calcul avec la suite des listes
calcul_cout_transport([A|R],[T|Q],C):- T#>0, C1#=C-T,
    calcul_cout_transport(R,Q,C1). %Si le lien est utilisé (Cij = 1),
    on vérifie si l'entrepôt peut servir le client (la valeur dans M n
    'est pas nulle), auquel cas on modifie la valeur de C et on
    continue par récurrence pour obtenir la valeur finale
```

— ***calcul_cout_fixe*** :

```
calcul_cout_fixe([],[],0). %Cas d'arrêt
calcul_cout_fixe([A|R],[0|Q],C):- calcul_cout_fixe(R,Q,C). %Si l'
    entrepôt n'est pas utilisé, on continue de parcourir les listes
calcul_cout_fixe([A|R],[T|Q],C):- C1#=C-A, calcul_cout_fixe(R,Q,C1).
    %Sinon, on ajoute la valeur des coûts fixes associés à l'entrepôt
    dans la boucle pour le reste du calcul par récurrence
```

On applique les deux prédicats définis précédemment aux variables souhaitées. Finalement, le programme d'optimisation s'écrit :

```
entrepots(C,Cout_total):-

C=[C11,C12,C13,
C21,C22,C23,
C31,C32,C33,
C41,C42,C43,
C51,C52,C53],

E=[18,10,28],

M=[5,4,0,0,3,
7,0,2,0,0,
0,5,5,4,8],

C::0..1,

C11+C12+C13#=1,
C21+C22+C23#=1,
C31+C32+C33#=1,
C41+C42+C43#=1,
C51+C52+C53#=1,


F1#=C11+C21+C31+C41+C51,
F2#=C12+C22+C32+C42+C52,
F3#=C13+C23+C33+C43+C53,

F=[F1,F2,F3],
```

```
calcul_cout_transport(C,M,Cout_transport),
calcul_cout_fixe(E,F,Cout_fixe),
Cout_total#=Cout_fixe+Cout_transport,
Cout_total#=<60,

minof(labeling(C),Cout_total), !.
```

Avec ce programme, on obtient le résultat suivant :

```
| ?- entrepots(C,C1).
C = [0,1,0,0,0,1,0,1,0,0,0,1,0,1,0]
C1 = 60
yes
```

La liste C donne la liste des relations utilisée pour servir chaque client. Par exemple, le client 1 est servi par le deuxième entrepôt. On peut remarquer que l'entrepôt 1 n'est pas utilisé. Le coût total obtenu vaut 60.

2 Problème de transport

On souhaite approvisionner 4 magasins en farine à partir d'un ensemble d'usines en minimisant les coûts de transport. À chaque magasin correspond une quantité (*demande*). À chaque usine correspond un stock (*capacité*). Il faut satisfaire la demande de chaque magasin sans dépasser la capacité de chaque usine.

Cette fois encore, on considère le schéma CSP.

— Définition des variables

```
L=[U1M1,U1M2,U1M3,U1M4,
U2M1,U2M2,U2M3,U2M4,
U3M1,U3M2,U3M3,U3M4],  %Coûts de transport entre usine et magasin

V=[7,8,5,19,
3,7,2,11,
9,5,10,18]  %Variable qui contient la valeur des coûts de transport
```

— Définition du domaine des variables

```
L::0..20  %Les coûts sont tous compris entre 0 et 20
```

— Définition des relations entre variables

```
%Respect des demandes des magasins
U1M1+U2M1+U3M1#=2,
U1M2+U2M2+U3M2#=2,
U1M3+U2M3+U3M3#=6,
U1M4+U2M4+U3M4#=3,

%Respect des stocks des usines
U1M1+U1M2+U1M3+U1M4#=4,
```

```
U2M1+U2M2+U2M3+U2M4#=6,
U3M1+U3M2+U3M3+U3M4#=3
```

— Définition de la fonction *calcul_cout*

```
calcul_cout([],[],0).
calcul_cout([0|R],[T|Q],C) :- calcul_cout(R,Q,C).
calcul_cout([A|R],[T|Q],C) :- C1#=C-A*T, calcul_cout(R,Q,C1).
```

On cherche à minimiser la variable correspondant au coût avec le code suivant :

```
calcul_cout(L,V,Cout_total),
minof(labeling(L),Cout_total), !.
```

Le résultat obtenu par l'algorithme est :

```
| ?- transport(L,C).
L = [0,0,4,0,0,2,2,2,2,0,0,1]
C = 96
yes
```

Le trajet optimal pour un coût de transport minimisé correspond à un coût de 96 où l'on récupère l'usine 1 livre 4 unités au magasin 3 par exemple.

3 Ordonnancement PERT

On cherche à ordonner des tâches qui ont chacun des durées.

3.1 Date de démarrage au plus tôt de la tâche J

— *Définition des variables*

```
Debut=[A,B,C,D,E,F,G,H,I,J],

Debut::0..100
```

Debut est la variable qui contient les dates de début de chacune des tâches. On fixe ces valeurs entre 0 et 100 avec 100 suffisamment grand pour que les valeurs trouvées correspondent.

— *Contraintes sur les variables*

```
A+5#=<B, A+5#=<C, A+5#=<D, %Contraintes pour A
B+4#=<E, %Contrainte pour B
C+3#=<F,C+3#=<G, %Contraintes pour C
D+2#=<F, %Contrainte pour D
E+1#=<H, %Contrainte pour E
F+5#=<I, %Contrainte pour F
G+4#=<I, %Contrainte pour G
H+3#=<J, %Contrainte pour H
I+2#=<J, %Contrainte pour I
```

On indique que le point de départ d'une tâche T2 qui doit avoir lieu après une tâche T1 ne peut être inférieur à T1 + sa durée respective. Sinon, cela signifierait que les tâches pourraient se chevaucher.

— *Condition d'optimisation*

Pour obtenir l'optimisation des débuts avec le début de la tâche J le plus tôt possible, on ajoute le code suivant :

```
minof(labeling(Debut),J),!.
```

— *Résultats*

Le programme précédent nous donne le résultat suivant :

```
| ?- pert(Debut).
Debut = [0,5,5,5,9,8,8,10,13,15]
yes
```

La tâche J commence donc au plus tôt sur le 15ème créneau.

3.2 Tâches qui ont une date flexible

Cette fois-ci, on fixe la date de démarrage de J à la valeur trouvée précédemment. On ajoute donc une contrainte :

```
J#=15.
```

On ne cherche plus à optimiser le problème mais à trouver toutes les solutions possibles. Ainsi, il n'est plus nécessaire d'utiliser la ligne d'optimisation précédente avec *minof*.

On obtient le résultat suivant :

```
| ?- pert(Debut).
Debut = [0,_03c8::[5..7],5,_0418::[5..6],_0440::[9..11],8,_0490::[8..9],
    _04b8::[10..12],13,15]
yes
```

Les tâches flexibles sont les tâches B, D, E, G et H données avec leurs dates de démarrage possibles respectives.

4 Ordonnancement disjonctif

On souhaite cette fois-ci que les tâches E, F et G ne se déroulent pas en même temps. On reprend les contraintes écrites pour l'exercice 3, question 1 et on ajoute un prédicat *pasenmemetemps*. Ce prédicat va vérifier que deux tâches associées à leur durée respective ne se déroulement pas en même temps.

La disjonction des trois tâches s'écrit alors :

```
pasenmemetemps(E,1,F,5),
pasenmemetemps(E,1,G,4),
pasenmemetemps(F,5,G,4)
```

Avec ces modifications, on obtient le résultat suivant :

```
| ?- pert(Debut).
Debut = [0,5,5,5,17,12,8,18,17,21]
yes
```

Les dates de début sont alors modifiées par rapport à précédemment mais on obtient bien les trois tâches E, F et G séparées.

5 JobShop avec ressources et sans bénéfice

Le but de cet exercice est de déterminer à quel moment dans le temps nous allons démarrer la tache numéro i. Pour cela, si deux tâches doivent être faites par la même machine, on contraint leur début de tâche grâce au prédicat *respect_duree*. Ce prédicat va parcourir l'ensemble de combinaison de tâche (Ti,Tj) avec i et j différents et sera vrai si Ti et Tj utilisent deux machines différentes, ou si elles utilisent les mêmes machines mais leur commencent de manière suffisamment espacée pour ne pas compromettre la terminaison de l'autre tâche.

On implémente l'exercice avec le code suivant :

```
machine(1,1). %(tache,machine)
machine(2,1).
machine(3,2).
machine(4,2).
machine(5,1).
machine(6,2).
machine(7,1).
machine(8,2).
duree(1,2). %(machine duree)
duree(2,3).
duree(3,1).
duree(4,2).
duree(5,3).
duree(6,4).
duree(7,2).
duree(8,3).

jobshop([De1,De2,De3,De4,De5,De6,De7,De8]):- % Dei contient la date de
    début de la tache i
[De1,De2,De3,De4,De5,De6,De7,De8]::0..12,
Ld=[De2,De3,De4,De5,De6,De7,De8],
Lt=[2,3,4,5,6,7,8],
respecte_duree(De1,1,Ld,Ld,Lt,Lt),
labeling([De1,De2,De3,De4,De5,De6,De7,De8]),!. % On se contente d une
    solution

respecte_duree(_Dei,_Ti,[],[],[],[]).
respecte_duree(_Dei,_Ti,[Dej|Qd],[],[Tj|Qt],[]):-respecte_duree(Dej,Tj,Qd,
    Qd,Qt,Qt).
respecte_duree(Dei,Ti,Ld,[Dej|Qd],Lt,[Tj|Qt]):-
(machine(Ti,M),machine(Tj,M),duree(Ti,Dui),duree(Tj,Duj),(Dej#>=Dei+Dui;
    Dei#>=Dej+Duj),respecte_duree(Dei,Ti,Ld,Qd,Lt,Qt))
```

```
;(machine(Ti,Mi),machine(Tj,Mj),Mi\=Mj,respecte_duree(Dei,Ti,Ld,Qd,Lt,Qt))
    .
```

On obtient les résultats suivants :

```
| ?- jobshop([De1,De2,De3,De4,De5,De6,De7,De8])
De1=0
De2=2
De3=0
De4=1
De5=5
De6=3
De7=8
De8=7
yes
```

Les variables Dei nous donnent les dates de début de chacune des tâches.

6 JobShop avec bénéfice

Le but de la première question est l'élaboration d'un plan pour produire un produit de chaque. Pour cela, on se propose de déterminer le début dans le temps de chaque tâche. On utilise deux contraintes :

— Une machine ne peut pas faire des tâches en même temps

— Pour un même produit, les tâches doivent être faites dans l'ordre

La résolution est effectuée à l'aide du code suivant :

```
benefice(1,5). % (produit,benefice)
benefice(2,3).
benefice(3,2).
duree(1,1,2). % (produit,tache,duree)
duree(1,2,4).
duree(1,3,3).
duree(2,1,3).
duree(2,2,2).
duree(3,1,1).
duree(3,2,3).
machine(1,1,2). % (produit,tache,machine)
machine(1,2,4).
machine(1,3,3).
machine(2,1,3).
machine(2,2,2).
machine(3,1,1).
machine(3,2,3).

% question 1
```

```
jobshopbenef([Duree,De11,De12,De13,De21,De22,De31,De32]):- % Deij contient
    la date de début de la tache j du produit i
[De11,De12,De13,De21,De22,De31,De32]::0..12,
De12#>=De11+2,De13#>=De12+4,De22#>=De21+3,De32#>=De31+1, % les taches
    doivent se faire dans l ordre pour un produit donne
(De12#>=De21+3;De21#>=De12+4), % on n utilise pas la machine 1 pour deux
    taches en meme temps
(De12#>=De32+3;De32#>=De12+4), % on n utilise pas la machine 1 pour deux
    taches en meme temps
(De32#>=De21+3;De21#>=De32+3), % on n utilise pas la machine 1 pour deux
    taches en meme temps
(De13#>=De22+2;De22#>=De13+3), % on n utilise pas la machine 2 pour deux
    taches en meme temps
(De11#>=De31+1;De31#>=De11+2), % on n utilise pas la machine 3 pour deux
    taches en meme temps
F11#=De11+2,
F12#=De12+4,
F13#=De13+3,
F21#=De21+3,
F22#=De22+2,
F31#=De31+1,
F32#=De32+3,
max([F11,F12,F13,F21,F22,F31,F32],Duree).

% il faut ensuite ecrire
% minof(jobshopbenef([Duree,De11,De12,De13,De21,De22,De31,De32]),Duree)
% afin de minimiser la durée

max([],R,R).
max([T|Q],M,R):- R#>=T,M#>=T,max(Q,M,R).
max([T|Q],M,R):- R#>=T,T#>M,max(Q,T,R).
max([T|Q],R):- max(Q,T,R),R#>=T.
```

On cherche ensuite une solution où la durée est minimale. L'exécution du code ci-dessous donne la réponse

```
| ?- minof(jobshopbenef([Duree,De11,De12,De13,De21,De22,De31,De32]),Duree)
Duree=10
De11=1
De12=3
De13=7
De21=0
De31=0
De32=7
yes
```

7 Problème d'emploi du temps

Le but étant d'associer des conférences d'une demi-journée à des créneaux en tenant compte de certaines incompatibilités entre les créneaux et le fait que seulement 3 conférences peuvent avoir lieu en même temps. Pour cela, nous numérotons les demi-journées de 1 à 4 et associons ces valeurs aux conférences de A à K en marquant également les contraintes. La contrainte qui indique qu'il ne peut pas avoir plus de 3 conférences en même temps est contenue dans le prédicat *salle_ dispo(L,X)*. Ce dernier va vérifier que le nombre d'apparitions de X dans L est inférieur ou égal à 3.

```
congres([A,B,C,D,E,F,G,H,I,J,K]):-
[A,B,C,D,E,F,G,H,I,J,K]::1..4 % selon une des 4 demis journee
,A#\=J, J#\=I, I#\=E, E#\=C, C#\=F, F#\=G, D#\=H, B#\=D, K#\=E,
B#\=I, H#\=G, B#\=H, B#\=G, I#\=H, I#\=G,
A#\=G, A#\=E, G#\=E,
B#\=H, B#\=K, H#\=K,
A#\=B, A#\=C ,A#\=H, B#\=C, B#\=H, C#\=H,
D#\=F, F#\=J, D#\=J,
J#>E, K#>D, K#>F,
salle_dispo([A,B,C,D,E,F,G,H,I,J,K],1), % Pas plus de 3 creneaux dans la
    premiere demi-journee
salle_dispo([A,B,C,D,E,F,G,H,I,J,K],2), % Pas plus de 3 creneaux dans la
    deuxième demi-journee
salle_dispo([A,B,C,D,E,F,G,H,I,J,K],3), % Pas plus de 3 creneaux dans la
    troisieme demi-journee
salle_dispo([A,B,C,D,E,F,G,H,I,J,K],4)  % Pas plus de 3 creneaux dans la
    derniere demi-journee
,labeling([A,B,C,D,E,F,G,H,I,J,K]),!. % on se contente dune seule solution

salle_dispo([],_X,C):- C#<3.
salle_dispo([X|Q],X,C):- C1 is (C+1),salle_dispo(Q,X,C1).
salle_dispo([_T|Q],X,C):- salle_dispo(Q,X,C).
salle_dispo(L,X):-salle_dispo(L,X,0).
```

La configuration suivante répond donc aux contraintes sans se soucier de l'attribution des salles :

```
| ?- congres([A,B,C,D,E,F,G,H,I,J,K])
A=1
B=2
C=3
D=1
E=2
F=2
G=3
H=4
I=1
J=4
K=3
yes
```

Le but maintenant est également d'attribuer les salles, le prédicat *congres2* s'en charge. Pour cela, nous reprenons les mêmes contraintes que congres auxquelles nous rajoutons le prédicat *affectsalles*. Celui-ci marche en deux temps, d'abord une affectation aléatoire des salles avec *affectsalle* puis avec le prédicat *verif*, on vérifie qu'une même salle n'est pas utilisée deux fois au même moment.

```
congres2([LA,LB,LC,LD,LE,LF,LG,LH,LI,LJ,LK]):- % Li=[numero de demie
    journee,numero de salle]
  [A,B,C,D,E,F,G,H,I,J,K]::1..4,
  salle_dispo([A,B,C,D,E,F,G,H,I,J,K],1), % Pas plus de 3 creneaux dans la
      premiere demi-journee
  salle_dispo([A,B,C,D,E,F,G,H,I,J,K],2), % Pas plus de 3 creneaux dans la
      deuxième demi-journee
  salle_dispo([A,B,C,D,E,F,G,H,I,J,K],3), % Pas plus de 3 creneaux dans la
      troisieme demi-journee
  salle_dispo([A,B,C,D,E,F,G,H,I,J,K],4), % Pas plus de 3 creneaux dans la
      derniere demi-journee
  dif(A,J),dif(J,I),dif(J,E),dif(E,C),dif(C,F),
  dif(F,G),dif(D,H),dif(B,D),dif(K,E),dif(B,I),
  dif(B,H),dif(B,G),dif(I,H),dif(I,G),dif(H,G),
  dif(A,G),dif(A,E),dif(G,E),
  dif(B,H),dif(B,K),dif(H,K),
  dif(A,B),dif(A,C),dif(A,H),dif(B,C),dif(B,H),dif(C,H),
  dif(D,F),dif(D,J),dif(F,J),
  E#<J,D#<K,F#<K,
  affectsalles([A,B,C,D,E,F,G,H,I,J,K],[LA,LB,LC,LD,LE,LF,LG,LH,LI,LJ,LK])
      , % affectation des salles
  labeling([LA,LB,LC,LD,LE,LF,LG,LH,LI,LJ,LK]),!. % on se contente d une
      seule solution qui marche


salle(a). % definition des 3 salles
salle(b).
salle(c).

affectsalles(L,Ls):-
  affectsalle(L,Ls), % affectation aleatoire de salles sans contraintes
  verif(Ls). % verification que la meme salle nest pas utilisee sur deux
      conferences ayant lieu en meme temps


affectsalle([],[]).
affectsalle([X|L],[[X,Xs]|L2]):- % on range le numero de demi journee et
    une salle au hasard dans la deuxieme liste
  salle(Xs),
  affectsalle(L,L2).

% verifie si X est membre de la liste L
membre(X,[X|_]).
membre(X,[_|L]):-
```

```
  membre(X,L).

verif([]).
verif([X|L]):-
  not(membre(X,L)), % verification qu'on ait pas deux fois le meme couple
      demi journee salle
   verif(L).
```

Nous obtenons donc :

```
| ?- congres2([LA,LB,LC,LD,LE,LF,LG,LH,LI,LJ,LK])
LA=[1,a]
LB=[2,a]
LC=[3,a]
LD=[1,b]
LE=[2,b]
LF=[2,c]
LG=[3,b]
LH=[4,a]
LI=[1,c]
LJ=[4,b]
LK=[3,c]
yes
```

8 Réunion

Le but ici est de trouver un créneau dans la semaine à la minute près qui permette à 4 personnes de se réunir dans une salle donnée. Le prédicat *disponible* relie chaque personne et la liste des intervalles de temps où cette personne est libre dans la semaine. Le prédicat *reunion* est chargé de trouver la date où tout le monde peut se réunir, pour cela les contraintes sont les suivantes :

— La date doit être dans la semaine (intervalle de définition de Date)

— Celle-ci doit convenir à tout le monde

Pour assurer cette dernière contrainte, nous avons créé le prédicat *convient* qui indique si la date convient à une personne donnée. Pour que cela soit le cas, il faut que le temps de la réunion [date,date+45] soit contenue dans un des intervalles de temps où la personne est disponible. C'est ce que gère le prédicat *contenue*.

```
disponible(jean,[[540,600],[810,855],[2115,2145],[2340,2460],
[3360,3495],[5040,5070],[6460,6435]]).
disponible(marie,[[600,630],[750,795],[2175,2265],
[2340,2460],[3420,3495],[4800,5130]]).
disponible(pierre,[[540,690],[750,900],[1980,2100],
[2160,2520],[3480,3555],[4800,4920],[5520,5610],[6300,6555]]).
disponible(jacques,[[630,900],[1920,2040],[1920,2040],
[2280,2400],[3360,3540],[3720,3825],[4905,4995]]).
disponible(salle,[[480,720],[2280,2400],
```

```
[3360,3420],[5040,5160],[6240,6360]]).

reunion(Date):-
  Date::0..7200, % domaine qui couvre les 5 jours LMMJV
  convient(jean,Date),
  convient(marie,Date),
  convient(pierre,Date),
  convient(jacques,Date),
  convient(salle,Date),
  labeling(Date),!. % on cherche une date qui marche

convient(X,Date):-disponible(X,L),contenue(Date,L).

contenue(Date,[[Debut,Fin]|_Q]):-Date#>=Debut,Date+45#=<Fin.
contenue(Date,[_C|Q]):-contenue(Date,Q).
```

Nous obtenons donc le résultat suivant :

```
| ?- reunion(Date)
Date=2340
yes
```

La réunion aura donc lieu mardi à 15h.

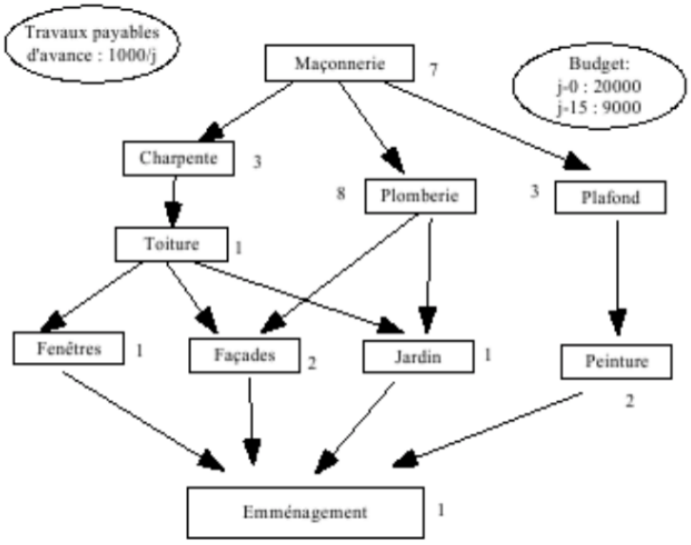
9 Planification

Le but de cet exercice est de planifier les taches à réaliser pour construire une maison. Dans un premier temps, nous ne nous attardons pas sur le budget, le but étant de faire les tâches le plus rapidement possible afin d'emménager le plus rapidement possible. Pour cela, le prédicat *maison* va se charger de donner le début de chaque tâche, il prend en compte 3 contraintes :

— La durée de chacune des tâches

— L'ordonnancement des tâches

— Minimiser la date de l'emménagement




```
%sans prise en compte du budget
maison([Dmaco,Dchar,Dplom,Dplaf,Dtoit,Dfene,Dfaca,Djard,Dpein,Demme]):-
[Dmaco,Dchar,Dplom,Dplaf,Dtoit,Dfene,Dfaca,Djard,Dpein,Demme]::0..30,
Lmaco#=7,Lchar#=3,Lplom#=8,Lplaf#=3,Ltoit#=1,
Lfene#=1,Lfaca#=2,Ljard#=1,Lpein#=2,Lemme#=1,
Fmaco#=Dmaco+Lmaco,Fchar#=Dchar+Lchar,Fplom#=Lplom+Dplom,Fplaf#=Dplaf+
    Lplaf,
Ftoit#=Dtoit+Ltoit,Ffene#=Dfene+Lfene,Ffaca#=Dfaca+Lfaca,Fjard#=Djard+
    Ljard,
Fpein#=Dpein+Lpein,Femme#=Demme+Lemme,
Dchar#>=Fmaco,Dplom#>=Fmaco,Dplaf#>=Fmaco,
Dtoit#>=Fchar,Dfene#>=Ftoit,Dfaca#>=Ftoit,Dfaca#>=Fplom,Djard#>=Ftoit,
    Djard#>=Fplom,
Dpein#>=Fplaf,
Demme#>=Ffene,Demme#>=Ffaca,Demme#>=Fjard,Demme#>=Fpein,
minof(labeling([Dmaco,Dchar,Dplom,Dplaf,Dtoit,Dfene,Dfaca,Djard,Dpein,
    Demme]),Femme).

%prise en compte du budget
maison2([Dmaco,Dchar,Dplom,Dplaf,Dtoit,Dfene,Dfaca,Djard,Dpein,Demme]):-
[Dmaco,Dchar,Dplom,Dplaf,Dtoit,Dfene,Dfaca,Djard,Dpein,Demme]::0..30,
Lmaco#=7,Lchar#=3,Lplom#=8,Lplaf#=3,Ltoit#=1,
Lfene#=1,Lfaca#=2,Ljard#=1,Lpein#=2,Lemme#=1,
Fmaco#=Dmaco+Lmaco,Fchar#=Dchar+Lchar,Fplom#=Lplom+Dplom,Fplaf#=Dplaf+
    Lplaf,
Ftoit#=Dtoit+Ltoit,Ffene#=Dfene+Lfene,Ffaca#=Dfaca+Lfaca,Fjard#=Djard+
    Ljard,
Fpein#=Dpein+Lpein,Femme#=Demme+Lemme,
Dchar#>=Fmaco,Dplom#>=Fmaco,Dplaf#>=Fmaco,
Dtoit#>=Fchar,Dfene#>=Ftoit,Dfaca#>=Ftoit,Dfaca#>=Fplom,Djard#>=Ftoit,
    Djard#>=Fplom,
Dpein#>=Fplaf,
```

```
Demme#>=Ffene ,Demme#>=Ffaca ,Demme#>=Fjard ,Demme#>=Fpein ,
calculbudget15([[Dmaco,Fmaco],[Dchar,Fchar],[Dplom,Fplom],[Dplaf,Fplaf],[
    Dtoit,Ftoit],
[Dfene,Ffene],[Dfaca,Ffaca],[Djard,Fjard],[Dpein,Fpein],[Demme,Femme]],0,
    Budget15),
Budget15#=20000, %on impose le bodget des 15 premiers jours
minof(labeling([Dmaco,Dchar,Dplom,Dplaf,Dtoit,Dfene,Dfaca,Djard,Dpein,
    Demme]),Demme),!.

calculbudget15([],Budget15,Budget15).
calculbudget15([[Xd,Xf]|Q],Acc,Budget15):- 15#>Xf,Acc1#=Acc+1000*(Xf-Xd),
    calculbudget15(Q,Acc1,Budget15).
calculbudget15([[Xd,Xf]|Q],Acc,Budget15):- Xf#>=15,Xd#<15,Acc1#=Acc
    +1000*(15-Xd),calculbudget15(Q,Acc1,Budget15).
calculbudget15([[Xd,Xf]|Q],Acc,Budget15):- Xf#>=15,Xd#>=15,calculbudget15(
    Q,Acc,Budget15).
```

Nous avons donc pour ce problème-là :

```
| ?- maison([Dmaco,Dchar,Dplom,Dplaf,Dtoit,Dfene,Dfaca,Djard,Dpein,Demme])
Dmaco=1
Dchar=8
Dplom=8
Dplaf=8
Dtoit=1
Dfene=2
Dfaca=16
Dpein=11
Demme=18
yes
```

10 Qui a gagné la médaille d'or ?

Pour cet exercice, nous avons rédigé le code qui nous semblait donner le résultat mais nous n'avons pas réussi à obtenir de réponse avec prolog.

```
whohaswon([Jean,Pierre,Paul,Andre,Roland]):-

%expliciter les variables
Jean=[Jbles,Jspor,Jbois,Jposi,Jmeda],
Pierre=[Pibles,Pispor,Pibois,Piposi,Pimeda],
Paul=[Pables,Paspor,Pabois,Paposi,Pameda],
Andre=[Ables,Aspor,Abois,Aposi,Ameda],
Roland=[Rbles,Rspor,Rbois,Rposi,Rmeda],

% definition de ce que l on sait deja
Pibles#=bras_casse,Paspor#=equitation,Abois#=the,

% definition des valeurs possibles
```

```
blessure(Jbles),blessure(Pables),blessure(Ables),blessure(Rbles),
sport(Jspor),sport(Pispor),sport(Aspor),sport(Rspor),
boisson(Jbois),boisson(Pibois),boisson(Pabois),boisson(Rbois),
position(Jposi),position(Piposi),position(Paposi),position(Aposi),position
    (Rposi),
medaille(Jmeda),medaille(Pimeda),medaille(Pameda),medaille(Ameda),medaille
    (Rmeda),

% deux a deux differents
different(Jean,[Pierre,Paul,Andre,Roland],[Pierre,Paul,Andre,Roland]),

% Paul a droite de Roland
Paposi#=Rposi+1,

% conditions imposees par l ennonce
deuxiemecondition([Jean,Pierre,Paul,Andre,Roland]),
quatriemecondition([Jean,Pierre,Paul,Andre,Roland]),
cinquiemecondition([Jean,Pierre,Paul,Andre,Roland]),
sixiemecondition([Jean,Pierre,Paul,Andre,Roland]),
huitiemecondition([Jean,Pierre,Paul,Andre,Roland]),
dixiemecondition([Jean,Pierre,Paul,Andre,Roland]),
onziemecondition([Jean,Pierre,Paul,Andre,Roland]),
douziemecondition(Jean,[Pierre,Paul,Andre,Roland],[Pierre,Paul,Andre,
    Roland]),
treiziemecondition(Jean,[Pierre,Paul,Andre,Roland],[Pierre,Paul,Andre,
    Roland]),
quatorziemecondition(Jean,[Pierre,Paul,Andre,Roland]).

% deux sportifs ont necessairement des attributs differents
different(_X,[],[]).
different(_X,[Xs|Qs],[]):-different(Xs,Qs,Qs).
different([Bles1,Spor1,Bois1,Posi1,Meda1],L,[[Bles2,Spor2,Bois2,Posi2,
    Meda2]|Q]):-
Bles1#\=Bles2,Spor1#\=Spor2,Bois1#\=Bois2,Posi1#\=Posi2,Meda1#\=Meda2,
different([Bles1,Spor1,Bois1,Posi1,Meda1],L,Q).

% celui qui s est fait une entorse a du abandoner
deuxiemecondition([[Bles,_Spor,_Bois,_Posi,Meda]|_Q]):-
Bles#=entorse,Meda#=abandon.
deuxiemecondition([_X|Q]):-deuxiemecondition(Q).

% le perchiste bois du cafe
quatriemecondition([[_Bles,Spor,Bois,_Posi,_Meda]|_Q]):-
Spor#=saut_a_la_perche,Bois#=cafe.
quatriemecondition([_X|Q]):-quatriemecondition(Q).

% celui qui boit du jus de fruit a fait une insolation
cinquiemecondition([[Bles,_Spor,Bois,_Posi,_Meda]|_Q]):-
Bois#=jus_de_fruit,Bles#=insolation.
```

```
cinquiemecondition([_X|Q]):-cinquiemecondition(Q).

% le decathlonien a un rhume
sixiemecondition([[Bles,Spor,_Bois,_Posi,_Meda]|_Q]):-
Spor#=decathlon,Bles#=rhume.
sixiemecondition([_X|Q]):-sixiemecondition(Q).

%celui qui bois de l eau a gagne la medaille d or
huitiemecondition([[_Bles,_Spor,Bois,_Posi,Meda]|_Q]):-
Bois#=eau,Meda#=or.
huitiemecondition([_X|Q]):-huitiemecondition(Q).

%celui du milieu fait de l escrime
dixiemecondition([[_Bles,Spor,_Bois,Posi,_Meda]|_Q]):-
Posi#=3,Spor#=escrime.
dixiemecondition([_X|Q]):-dixiemecondition(Q).

% le premier a gauche a une cocarde a l oeil
onziemecondition([[Bles,_Spor,_Bois,Posi,_Meda]|_Q]):-
Posi#=1,Bles#=cocarde.
onziemecondition([_X|Q]):-onziemecondition(Q).

%buveur de lait a cote de celui qui finit 4eme
douziemecondition(_X,[Xs|Qs],[]):-douziemeconditiont(Xs,Qs,Qs).
douziemecondition([_Bles1,_Spor1,Bois1,Posi1,Meda1],_L,[[_Bles2,_Spor2,
    Bois2,Posi2,Meda2]|_Q]):-
(Bois1#=lait,Meda2#=quatrieme,(Posi2#=Posi1+1;Posi1#=Posi2+1));(Bois2#=
    lait,Meda1#=quatrieme,(Posi2#=Posi1+1;Posi1#=Posi2+1)).
douziemecondition([Bles1,Spor1,Bois1,Posi1,Meda1],L,[_X|Q]):-
douziemecondition([Bles1,Spor1,Bois1,Posi1,Meda1],L,Q).

%buveur de the a cote de celui qui a eu une medaille de bronze
treiziemecondition(_X,[Xs|Qs],[]):-treiziemecondition(Xs,Qs,Qs).
treiziemecondition([_Bles1,_Spor1,Bois1,Posi1,Meda1],_L,[[_Bles2,_Spor2,
    Bois2,Posi2,Meda2]|_Q]):-
(Bois1#=the,Meda2#=bronze,(Posi2#=Posi1+1;Posi1#=Posi2+1));(Bois2#=the,
    Meda1#=bronze,(Posi2#=Posi1+1;Posi1#=Posi2+1)).
treiziemecondition([Bles1,Spor1,Bois1,Posi1,Meda1],L,[_X|Q]):-
treiziemecondition([Bles1,Spor1,Bois1,Posi1,Meda1],L,Q).

% celui qui a la cocarde a l oeil est a cote de Jean
quatorziemecondition([_Jbles,_Jspor,_Jbois,Jposi,_Jmeda],[[Bles,_Spor,
    _Bois,Posi,_Meda]|_Q]):-
Bles#=cocarde,(Jposi#=Posi+1;Posi#=Jposi+1).
quatorziemecondition(Jean,[_X|Q]):-quatorziemecondition(Jean,Q).

blessure(entorse).
blessure(insolation).
blessure(rhume).
```

```
blessure(cocarde).

sport(saut_a_la_perche).
sport(decathlon).
sport(escrime).
sport(boxe).

boisson(cafe).
boisson(jus_de_fruit).
boisson(eau).
boisson(lait).

position(1).
position(2).
position(3).
position(4).
position(5).

medaille(or).
medaille(argent).
medaille(bronze).
medaille(quatrieme).
medaille(abandon).
```