Esercizi proposti

1. Implementare il proprio "albero genealogico", definendo (mediante fatti) la relazione **genitore** (fino ai bisnonni, con tutti gli zii e i cugini). Definire poi i predicati a due argomenti: **fratello** (che rappresenti sia la relazione fratello che sorella), **nonno** (nonno o nonna), **zio** (zio o zia), **cugino** (cugino o cugina) e **discendente**.

2. Definire un predicato **fact(+X,?Y)**, vero se Y è il fattoriale di X.

3. Definire il predicato **palindroma(X)**, vero se X è una lista palindroma (se la lista viene letta in un verso o nell'altro si ottiene la stessa sequenza di elementi). Ad esempio `[a,b,c,b,a]` è palindroma, `[a,b,c,a]` non lo è.

4. Definire un predicato **maxlist(+L,?N)** (dove L è una lista di numeri), vero se N e' il massimo elemento della lista L. Fallisce se L è vuota.

5. Avendo definito

   ```
   pari(X) :- 0 is X mod 2.
   ```

   definire il predicato **split(+L,?P,?D)** = se L è una lista di interi, P è la lista contenente tutti gli elementi pari di L e D tutti quelli dispari (nello stesso ordine in cui occorrono in L).

6. Scrivere un programma che risolva il problema delle torri di Hanoi: il predicato principale **hanoi(N)** risolve il problema con N torri, spostando N dischi dal piolo A al piolo B, usando C come appoggio. Per farlo, richiama il predicato a 4 argomenti **hanoi(N,Start,Goal,Appoggio)** che sposta N dischi dal piolo Start al piolo Goal, usando Appoggio come piolo ausiliario. Lo spostamento di un singolo disco dal piolo Start al piolo Goal è rappresentato dalla stampa su video del messaggio: "Sposto un disco da Start a Goal".

   Ad esempio si avrà:

   ```
   ?- hanoi(3).
   Sposto un disco da A a B
   Sposto un disco da A a C
   Sposto un disco da B a C
   Sposto un disco da A a B
   Sposto un disco da C a A
   Sposto un disco da C a B
   Sposto un disco da A a B
   ```

   Per la stampa del messaggio, si vedano i predicati predefiniti **write** e **writeln**. Può essere inoltre utile utilizzare il predicato predefinito **atomic_list_concat**, che consente di costruire la concatenazione degli atomi in una stringa. Ad esempio:

```
?- Start='A', Goal='B',
   atomic_list_concat(['Sposto un disco da',Start,'a',Goal],' ',Out).
Start = 'A',
Goal = 'B',
Out = 'Sposto un disco da A a B'.
```

7. Definire un predicato **prefisso(Pre,L)** = la lista Pre è un prefisso della lista L. Ad esempio, i prefissi della lista `[1,2,3]` sono: la lista vuota `[]` e le liste `[1]`, `[1,2]` e `[1,2,3]` stessa.

8. Definire un predicato **suffisso(Suf,L)** = la lista Suf è un suffisso della lista L. Ad esempio, i suffissi della lista `[1,2,3]` sono: la lista vuota `[]` e le liste `[3]`, `[2,3]` e `[1,2,3]` stessa.

9. Definire un predicato **sublist(S,L)** = S è una sottolista di L costituita da elementi contigui in L. Ad esempio, le sottoliste di `[1,2,3]` sono: la lista vuota `[]` e le liste `[1]`, `[2]`, `[3]`, `[1,2]`, `[2,3]` e `[1,2,3]` stessa.

10. Definire i seguenti predicati:

    (a) **subset(+Sub,?Set)** = tutti gli elementi di Sub sono anche elementi di Set.

    (b) **rev(+X,?Y)** = Y è la lista che contiene gli stessi elementi di X, ma in ordine inverso.

    (c) **del_first(+X,+L,?Resto)** = Resto è la lista che si ottiene da L cancellando la prima occorrenza di X. Fallisce se X non occorre in L.

    Attenzione: se L contiene più occorrenze di X, il backtracking non deve fornire altre soluzioni (cancellando la seconda occorrenza, la terza ecc.). Usare quindi opportunamente il cut o il not.

    (d) **del(+X,+L,?Resto)** = Resto è la lista che si ottiene da L cancellando tutte le occorrenze di X. Se X non occorre in L, Resto è uguale a L stessa.

    Attenzione: il backtracking non deve generare altre soluzioni, in cui alcune occorrenze X rimangono nella soluzione.

    (e) **subst(+X,+Y,+L,-Nuova)** = Nuova è la lista che si ottiene da L sostituendo tutte le occorrenze di X con Y. Se X non occorre in L, Nuova è uguale a L stessa.

    (f) **mkset(+L,-Set)** = Set è una lista senza ripetizioni che contiene tutti e solo gli elementi di L (senza utilizzare il predicato predefinito **list_to_set/2**).

    (g) **union(+A,+B,-Union)** = Union è una lista (senza ripetizioni, se anche A e B sono senza ripetizioni) che rappresenta l'unione di A e B.

11. Definire un predicato **cartprod(+A,+B,-Set)**, vero se A e B sono liste e Set una lista di coppie che rappresenta il prodotto cartesiano di A e B. Ad esempio:

```
?- cartprod([a,b,c],[1,2],Set).
Set = [ (a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)].
```

12. Definire un predicato **insert(X,L1,L2)**, vero se L2 si ottiene inserendo X in L1 (in qualsiasi posizione). Almeno una delle due liste L1 e L2 devono essere istanziate. Ad esempio:

```
?- insert(a,[1,2],X).
X = [a, 1, 2] ;
X = [1, a, 2] ;
X = [1, 2, a] ;
false.

?- insert(a,X,[1,a,2]).
X = [1, 2] ;
false.
```

13. Definire un predicato **permut(X,Y)** vero se X e Y sono liste e Y è una permutazione di X (senza usare il predicato predefinito **permutation/2**). Ad esempio:

```
?- permut([1,2,3],Permut).
Permut = [1, 2, 3] ;
Permut = [2, 1, 3] ;
Permut = [2, 3, 1] ;
Permut = [1, 3, 2] ;
Permut = [3, 1, 2] ;
Permut = [3, 2, 1] ;
false.
```

14. Definire un predicato **search_subset(+IntList,+N,?Set)**, dove IntList è una lista di interi positivi e N un intero positivo, che sia vero se Set è una lista rappresentante un sottoinsieme di IntList, tale che la somma degli elementi in Set è uguale a N. Si può assumere che IntList sia senza ripetizioni. Ad esempio:

```
?- search_subset([4,8,5,3,9,6,7],9,Subset).
Subset = [4, 5] ;
Subset = [3, 6] ;
Subset = [9] ;
false.
```

15. Conveniamo di rappresentare gli alberi binari usando l'atomo **empty** per l'albero vuoto e strutture della forma **t(Root,Left,Right)** per alberi con radice Root, sottoalbero sinistro Left e sottoalbero destro Right. Definire i predicati:

(a) **height(+T,?N)** = N è l'altezza dell'albero T. Il predicato fallisce se T è l'albero vuoto.
(b) **reflect(T,T1)** = T è l'immagine riflessa di T1. Almeno uno tra T e T1 devono essere completamente istanziati.
(c) **size(+T,?N)** = N è il numero di nodi dell'albero T.
(d) **labels(+T,-L)** = L è una lista di tutte le etichette dei nodi di T. Se diversi nodi di T hanno la stessa etichetta, la lista L conterrà ripetizioni dello stesso elemento. Gli elementi di L possono occorrere in qualsiasi ordine.
(e) **branch(+T,?Leaf,?Path)** = Path è una lista che rappresenta un ramo dalla radice di T fino a una foglia etichettata da Leaf.

16. Un grafo si può rappresentare mediante un insieme di fatti della forma **arc(X,Y)**, che definiscono la relazione binaria "esiste un arco da X a Y". Ad esempio:

```
arc(a,b).                           arc(c,d).
arc(a,e).                           arc(d,c).
arc(b,a).                           arc(d,b).
arc(b,c).                           arc(e,c).
arc(c,c).
```

Definire un predicato **path(?Start,?Goal,?Path)** = Path è una lista che rappresenta un cammino da Start a Goal nel grafo definito nel programma.

Suggerimento: utilizzare un predicato ausiliario a quattro argomenti **path(?Start,?Goal,?Path,+Visited)** = Path è una lista che rappresenta un cammino da Start a Goal che non passa per nessuno dei nodi della lista Visited.

17. Rappresentiamo le formule della logica proposizionale classica mediante strutture costruite mediante gli operatori definiti come segue:

```
:- op(600,yfx,&).     % Congiunzione
:- op(650,yfx,v).     % Disgiunzione
:- op(670,yfx,=>).    % Implicazione
:- op(680,yfx,<=>).   % Doppia implicazione
```

La negazione è rappresentata dall'operatore `-` (unario), che ha precedenza 200 e tipo fy.

(a) Definire un predicato a due argomenti `nnf(+X,-Y)` che, data una formula `X`, costruisca in `Y` la sua forma normale negativa.
(b) Rappresentiamo un'interpretazione mediante una lista di atomi: tutti e solo quelli veri nell'interpretazione. Definire un predicato `holds(+F,+L)`, che determini se la formula `F` è vera nell'interpretazione rappresentata da `L`.