Esercizio 1

Supponendo di avere una base di fatti *Prolog* relativa ad aule, docenti, corsi, orario delle lezioni e prerequisiti d'esame, ad esempio:

```
aula(n1, 220).
aula(n2, 180).
...
docente(rossi, 5467).
docente(bianchi, 5687).
...
corso(geometria, rossi).
corso(algebra, bianchi).
...
orario(geometria, lun, 1, n8).
orario(geometria, mer, 4, n2).
orario(algebra, ven, 7, v1).
...
prerequisito('analisi 1', 'analisi 2').
prerequisito('calcolatori A', 'calcolatori B').
...
```

definire le regole *Prolog* per i seguenti predicati:

a) `docenteOccupato(D, G)`

in cui il docente `D` ha almeno un'ora di lezione il giorno `G`;

b) `docenteLibero(D, G)`

in cui il docente `D` non ha lezione il giorno `G`;

c) `precedente(C, P)`

in cui `P` è un corso (direttamente o indirettamente) precedente al corso `C`.

Esercizio 1

```
aula(n1, 220).
aula(n2, 180).
...
docente(rossi, 5467).
docente(bianchi, 5687).
...
corso(geometria, rossi).
corso(algebra, bianchi).
...
orario(geometria, lun, 1, n8).
orario(geometria, mer, 4, n2).
orario(algebra, ven, 7, v1).
...
prerequisito('analisi 1', 'analisi 2').
prerequisito('calcolatori A', 'calcolatori B').
...
```

```
docenteOccupato(D,G) :- orario(C,G,_,_), corso(C,D).
```

```
docenteLibero(D,G) :- docente(D,_),
                      \+(docenteOccupato(D,G)).
```

```
precedente(C,P) :- prerequisito(P,C).
precedente(C,P) :- prerequisito(X,C), precedente(X,P).
```

Esercizio 2

Supponendo di avere una base di fatti *Prolog* relativa alla specifica di una relazione genitore-figlio, come nel seguente esempio,

```
genitore(guido, elena).
genitore(guido, luisa).
genitore(elena, giovanni).
genitore(elena, paola).
genitore(luisa, andrea).
genitore(luisa, dario).
...
```

specificare in *Prolog* i seguenti predicati:

- `fratello(X, Y)`: `X` è fratello di `Y`;
- `cugino(X, Y)`: `X` è cugino di `Y`;

Esercizio 2

Supponendo di avere una base di fatti *Prolog* relativa alla specifica di una relazione genitore-figlio, come nel seguente esempio,

```
genitore(guido, elena).
genitore(guido, luisa).
genitore(elena, giovanni).
genitore(elena, paola).
genitore(luisa, andrea).
genitore(luisa, dario).
...
```

specificare in *Prolog* i seguenti predicati:

- `fratello(X, Y)`: X è fratello di Y;
- `cugino(X, Y)`: X è cugino di Y;

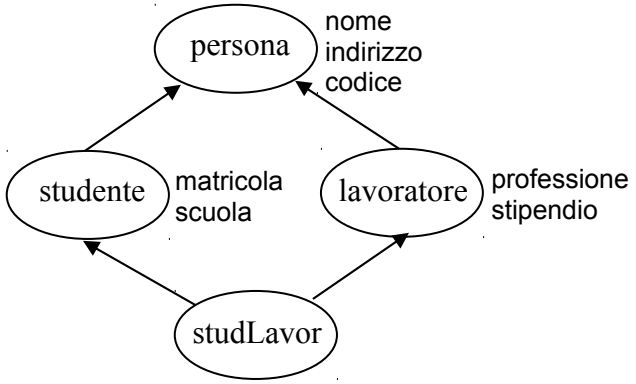
```
fratello(X, Y) :- genitore(Z, X), genitore(Z, Y), X \= Y.
```

```
cugino(X, Y) :- genitore(Z, X), genitore(W, Y), fratello(Z, W).
```

Esercizio 3

Data una base di fatti *Prolog* relativa alla descrizione di classi di un linguaggio ad oggetti, come nel seguente esempio,

```
class(persona).
var(nome, persona).
var(indirizzo, persona).
var(codice, persona).
class(studente).
inherits(studente, persona).
var(matricola, studente).
var(scuola, studente).
class(lavoratore).
inherits(lavoratore, persona).
var(professione, lavoratore).
var(stipendio, lavoratore).
class(studLavor).
inherits(studLavor, studente).
inherits(studLavor, lavoratore).
```



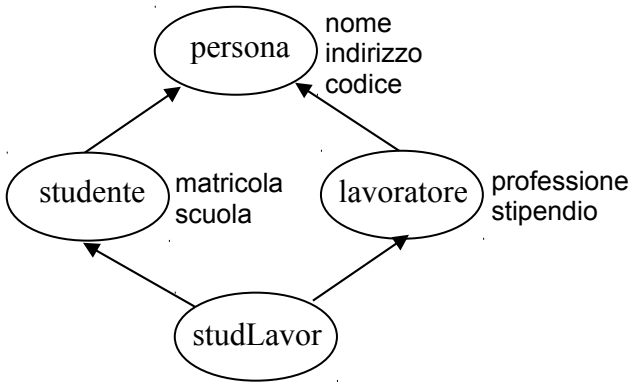
definire le regole Prolog per i seguenti predicati:

a) `superclass(C1, C2)`
   in cui `C1` è (direttamente o indirettamente) una superclasse di `C2`;

b) `ambref(V, C)`
   in cui `V` è una variabile definita in `C` o ereditata da una superclasse di `C`.

Esercizio 3

```
class(persona).
var(nome, persona).
var(indirizzo, persona).
var(codice, persona).
class(studente).
inherits(studente, persona).
var(matricola, studente).
var(scuola, studente).
class(lavoratore).
inherits(lavoratore, persona).
var(professione, lavoratore).
var(stipendio, lavoratore).
class(studLavor).
inherits(studLavor, studente).
inherits(studLavor, lavoratore).
```



definire le regole Prolog per i seguenti predicati:

a) `superclass(C1, C2)`
   in cui `C1` è (direttamente o indirettamente) una superclasse di `C2`;

b) `ambref(V, C)`
   in cui `V` è una variabile definita in `C` o ereditata da una superclasse di `C`.

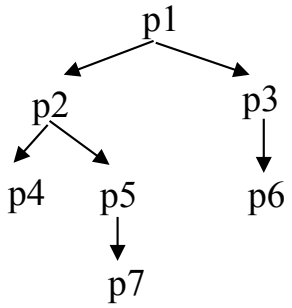
```
superclass(C1, C2) :- inherits(C2, C1).
superclass(C1, C2) :- inherits(C, C1), superclass(C, C2).
```

```
ambref(V, C) :- var(V, C).
ambref(V, C) :- var(V, C1), superclass(C1, C).
```

Esercizio 4

Data una base di fatti *Prolog* relativa alla descrizione di chiamate di procedure in un linguaggio imperativo, come nel seguente esempio,

```
calls(p1, p2).
calls(p1, p3).
calls(p2, p4).
calls(p2, p5).
calls(p3, p6).
calls(p5, p7).
```

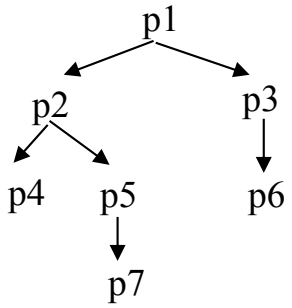


definire il predicato `activates(P, Q)`, in cui la procedura `Q` è (direttamente o indirettamente) chiamata dalla procedura `P`.

Esercizio 4

Data una base di fatti *Prolog* relativa alla descrizione di chiamate di procedure in un linguaggio imperativo, come nel seguente esempio,

```
calls(p1, p2).
calls(p1, p3).
calls(p2, p4).
calls(p2, p5).
calls(p3, p6).
calls(p5, p7).
```



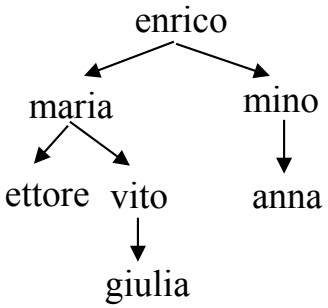
definire il predicato `activates(P, Q)`, in cui la procedura `Q` è (direttamente o indirettamente) chiamata dalla procedura `P`.

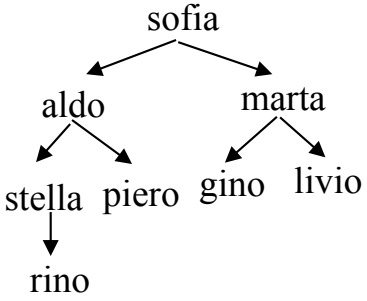
```
activates(P, Q) :- calls(P, Q).
activates(P, Q) :- calls(P, R), activates(R, Q).
```

Esercizio 5

Data una base di fatti *Prolog* relativa ad alberi genealogici, come nel seguente esempio:

```
genitore(enrico, maria).
genitore(enrico, mino).
genitore(maria, ettore).
...
genitore(sofia, aldo).
genitore(sofia, marta).
genitore(aldo, stella).
...
```



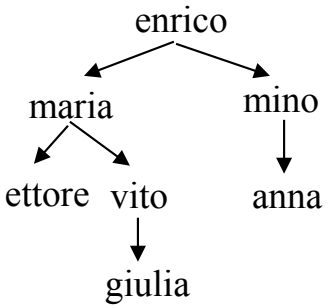


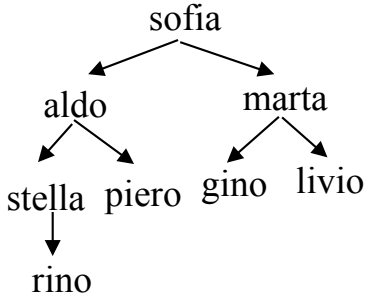
definire il predicato `parenti(X, Y)`, che stabilisce se `X` ed `Y` appartengono allo stesso albero genealogico.

Esercizio 5

Data una base di fatti *Prolog* relativa ad alberi genealogici, come nel seguente esempio:

```
genitore(enrico, maria).
genitore(enrico, mino).
genitore(maria, ettore).
...
genitore(sofia, aldo).
genitore(sofia, marta).
genitore(aldo, stella).
...
```





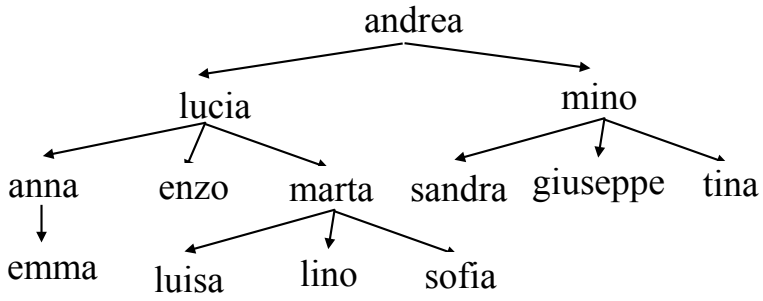
definire il predicato `parenti(X, Y)`, che stabilisce se `X` ed `Y` appartengono allo stesso albero genealogico.

```
antenato(X, X).
antenato(X, Y) :- genitore(X, Z), antenato(Z, Y).
parenti(X, Y) :- antenato(Z, X), antenato(Z, Y).
```

Esercizio 6

Data una base di fatti Prolog relativa alla relazione parentale tra un insieme di persone, come nel seguente esempio:

```
genitore(andrea, [lucia, mino]).
genitore(lucia, [anna, enzo, marta]).
genitore(mino, [sandra, giuseppe, tina]).
genitore(anna, [emma]).
genitore(marta, [luisa, lino, sofia]).
...
```



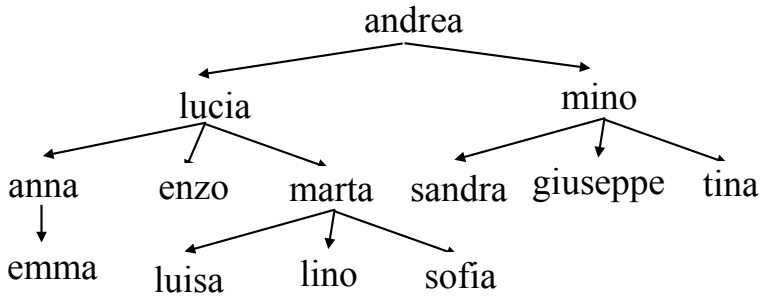
definire le regole per i seguenti predicati:

- `nonno(X, Y)`
  in cui `X` è nonno (o nonna) di `Y`;
- `antenato(X, Y)`
  in cui `X` è un antenato di `Y`.

Esercizio 6

Data una base di fatti Prolog relativa alla relazione parentale tra un insieme di persone, come nel seguente esempio:

```
genitore(andrea, [lucia, mino]).
genitore(lucia, [anna, enzo, marta]).
genitore(mino, [sandra, giuseppe, tina]).
genitore(anna, [emma]).
genitore(marta, [luisa, lino, sofia]).
...
```



definire le regole per i seguenti predicati:

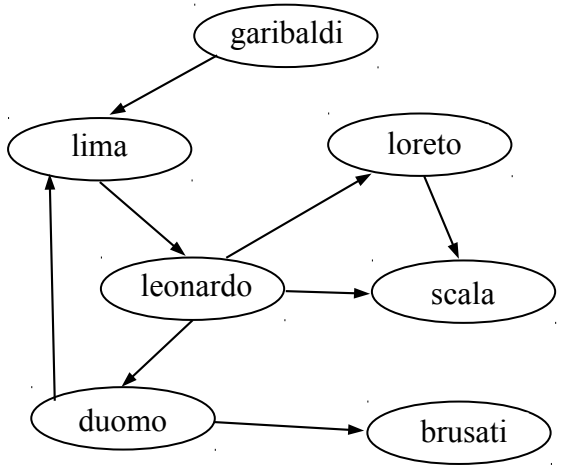
- `nonno(X, Y)`
  in cui `X` è nonno (o nonna) di `Y`;
- `antenato(X, Y)`
  in cui `X` è un antenato di `Y`.

```
nonno(X, Y) :- genitore(X, L1), member(Z, L1), genitore(Z, L2), member(Y, L2).

antenato(X, Y) :- genitore(X, L), member(Y, L).
antenato(X, Y) :- genitore(X, L), member(Z, L), antenato(Z, Y).
```

Esercizio 7

Data una base di fatti *Prolog* relativa alla specifica dei collegamenti tra le piazze di una città mediante vicoli a senso unico (percorribili in un'unica direzione), come nel seguente esempio:

```
vicolo(garibaldi, lima).
vicolo(lima, leonardo).
vicolo(leonardo, loreto).
vicolo(leonardo, scala).
vicolo(leonardo, duomo).
vicolo(loreto, scala).
vicolo(duomo, lima).
vicolo(duomo, brusati).
```



definire le regole per i seguenti predicati:

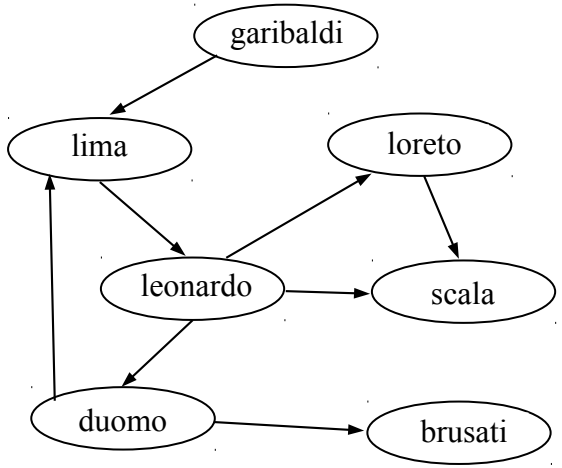
a) `raggiungibile(P1, P2)`
in cui la piazza `P2` è raggiungibile dalla piazza `P1` percorrendo una serie (non vuota) di vicoli;

b) `circolare(P)`
in cui `P` è una piazza che si trova su un tragitto circolare di vicoli.

Esercizio 7

Data una base di fatti *Prolog* relativa alla specifica dei collegamenti tra le piazze di una città mediante vicoli a senso unico (percorribili in un'unica direzione), come nel seguente esempio:

```
vicolo(garibaldi, lima).
vicolo(lima, leonardo).
vicolo(leonardo, loreto).
vicolo(leonardo, scala).
vicolo(leonardo, duomo).
vicolo(loreto, scala).
vicolo(duomo, lima).
vicolo(duomo, brusati).
```



definire le regole per i seguenti predicati:

a) `raggiungibile(P1, P2)`
in cui la piazza `P2` è raggiungibile dalla piazza `P1` percorrendo una serie (non vuota) di vicoli;

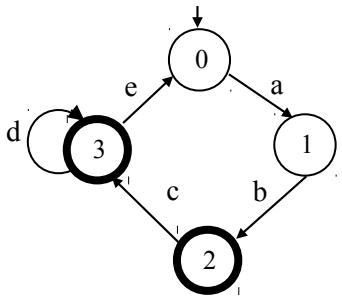
b) `circolare(P)`
in cui `P` è una piazza che si trova su un tragitto circolare di vicoli.

```
raggiungibile(P1, P2) :- vicolo(P1, P2).
raggiungibile(P1, P2) :- vicolo(P1, P), raggiungibile(P, P2).

circolare(P) :- raggiungibile(P, P).
```

Esercizio 8

Data una base di fatti *Prolog* relativa alla specifica di un automa, come nel seguente esempio:

```
state(0).
state(1).
state(2).
state(3).
initial(0).
final(2).
final(3).
trans(0,a,1).
trans(1,b,2).
trans(2,c,3).
trans(3,d,3).
trans(3,e,0).
```



S	L	gen(S, L)	generates(L)
1	[b]	true	false
1	[b,c]	true	false
2	[c,d,d]	true	false
2	[c,d,e]	false	false
2	[c,e,a,b]	true	false
3	[]	true	false
2	[a,b]	false	true
0	[a,b,c,d]	true	true

in cui l'automa ha uno ed un solo stato iniziale ed almeno uno stato finale, definire le regole per i seguenti predicati:

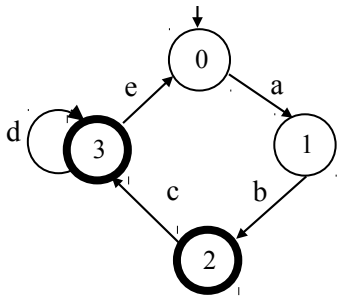
a) `gen(S, L)`
   in cui la lista (eventualmente vuota) `L` di simboli è generata da un cammino (eventualmente vuoto) che parte dallo stato `S` e termina in uno stato finale;

b) `generates(L)`
   in cui la lista `L` è generata da un cammino che parte dallo stato iniziale e termina in uno stato finale.

Esercizio 8

Data una base di fatti *Prolog* relativa alla specifica di un automa, come nel seguente esempio:

```
state(0).
state(1).
state(2).
state(3).
initial(0).
final(2).
final(3).
trans(0,a,1).
trans(1,b,2).
trans(2,c,3).
trans(3,d,3).
trans(3,e,0).
```



S	L	gen(S, L)	generates(L)
1	[b]	true	false
1	[b,c]	true	false
2	[c,d,d]	true	false
2	[c,d,e]	false	false
2	[c,e,a,b]	true	false
3	[]	true	false
2	[a,b]	false	true
0	[a,b,c,d]	true	true

in cui l'automa ha uno ed un solo stato iniziale ed almeno uno stato finale, definire le regole per i seguenti predicati:

a) `gen(S, L)`
in cui la lista (eventualmente vuota) `L` di simboli è generata da un cammino (eventualmente vuoto) che parte dallo stato `S` e termina in uno stato finale;

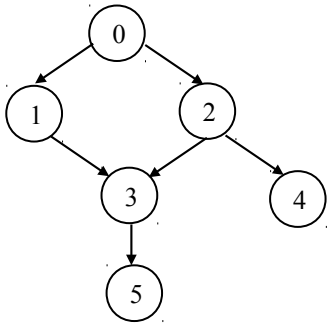
b) `generates(L)`
in cui la lista `L` è generata da un cammino che parte dallo stato iniziale e termina in uno stato finale.

```
gen(S, []) :- final(S).
gen(S, [X|Y]) :- trans(S,X,S2), gen(S2,Y).

generates(L) :- initial(S0), gen(S0, L).
```

Esercizio 9

È data una base di fatti *Prolog* relativa alla specifica di un grafo connesso, aciclico e diretto, come nel seguente esempio:

```
node(0, [1,2]).
node(1, [3]).
node(2, [3,4]).
node(3, [5]).
node(4, []).
node(5, []).
```



Ogni nodo è rappresentato dal suo identificatore e dalla lista di nodi successivi. Il grafo stabilisce un ordinamento temporale parziale tra eventi. Ad esempio, l'evento 0 accade prima di ogni altro evento. Gli eventi 1 e 2 accadono dopo l'evento 0. L'evento 3 accade dopo gli eventi 1 e 2. L'evento 1 accade prima dell'evento 5. D'altra parte, alcuni eventi non sono confrontabili. Ad esempio, non è possibile stabilire una relazione d'ordine tra gli eventi 1 e 2, nemmeno tra 4 e 5. Per definizione, ogni evento è confrontabile con se stesso. In sintesi, due eventi possono essere o meno confrontabili.
Sulla base di tale interpretazione del grafo, si chiede di specificare la seguente regola:

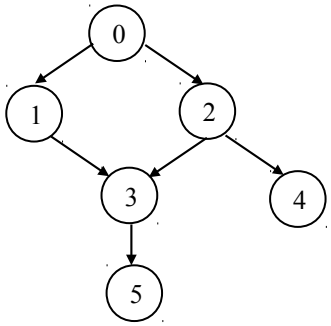
```
confrontabili(N1, N2)
```

in cui gli eventi `N1` ed `N2` sono confrontabili.

Esercizio 9

È data una base di fatti *Prolog* relativa alla specifica di un grafo connesso, aciclico e diretto, come nel seguente esempio:

```
node(0, [1,2]).
node(1, [3]).
node(2, [3,4]).
node(3, [5]).
node(4, []).
node(5, []).
```



Ogni nodo è rappresentato dal suo identificatore e dalla lista di nodi successivi. Il grafo stabilisce un ordinamento temporale parziale tra eventi. Ad esempio, l'evento 0 accade prima di ogni altro evento. Gli eventi 1 e 2 accadono dopo l'evento 0. L'evento 3 accade dopo gli eventi 1 e 2. L'evento 1 accade prima dell'evento 5. D'altra parte, alcuni eventi non sono confrontabili. Ad esempio, non è possibile stabilire una relazione d'ordine tra gli eventi 1 e 2, nemmeno tra 4 e 5. Per definizione, ogni evento è confrontabile con se stesso. In sintesi, due eventi possono essere o meno confrontabili.
Sulla base di tale interpretazione del grafo, si chiede di specificare la seguente regola:

`confrontabili(N1, N2)`

in cui gli eventi `N1` ed `N2` sono confrontabili.

```
link(N1, N2) :- node(N1, F), node(N2, _), member(N2, F).
collegato(N1, N2) :- link(N1, N2).
collegato(N1, N2) :- link(N1, N), collegato(N, N2).
confrontabili(N1, N2) :- node(N1, _), N1 = N2.
confrontabili(N1, N2) :- collegato(N1, N2).
confrontabili(N1, N2) :- collegato(N2, N1).
```

Esercizio 10

Specificare in *Prolog* il predicato `fib(N, F)`, che risulta vero qualora `F` sia il valore della funzione di Fibonacci, definita (matematicamente) nel seguente modo:

$$\text{fib}(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ \text{fib}(n\text{-}2) + \text{fib}(n\text{-}1) & \text{altrimenti.} \end{cases}$$

Esercizio 10

Specificare in *Prolog* il predicato `fib(N, F)`, che risulta vero qualora `F` sia il valore della funzione di Fibonacci, definita (matematicamente) nel seguente modo:

$$\text{fib}(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ \text{fib}(n\text{-}2) + \text{fib}(n\text{-}1) & \text{altrimenti.} \end{cases}$$

```
fib(0,0) :- !.
fib(1,1) :- !.
fib(N,F) :- N1 is N-1, N2 is N-2,
            fib(N1,F1), fib(N2,F2),
            F is F1+F2.
```

Esercizio 11

Definire nel linguaggio logico *Prolog* il predicato `shrunk(L,S)`, che risulta vero quando `S` rappresenta la lista degli elementi di `L` in posizione dispari.

Esercizio 11

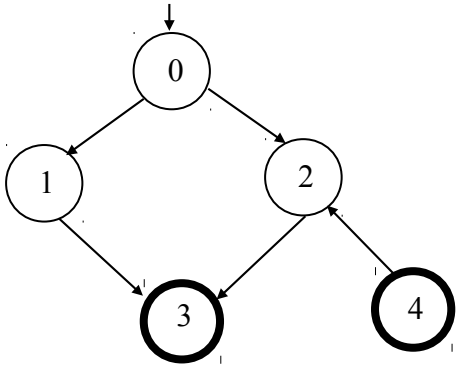
Definire nel linguaggio logico *Prolog* il predicato `shrunk(L,S)`, che risulta vero quando `S` rappresenta la lista degli elementi di `L` in posizione dispari.

```
shrunk([],[]).
shrunk([X],[X]).
shrunk([X, _ | T], [X | S]) :- shrunk(T, S).
```

Esercizio 12

È data una base di fatti *Prolog* relativa alla specifica di un grafo aciclico diretto, come nel seguente esempio,

```
stati([0,1,2,3,4]).
iniziale(0).
finali([3,4]).
arco(0,1).
arco(0,2).
arco(1,3).
arco(2,3).
arco(4,2).
```

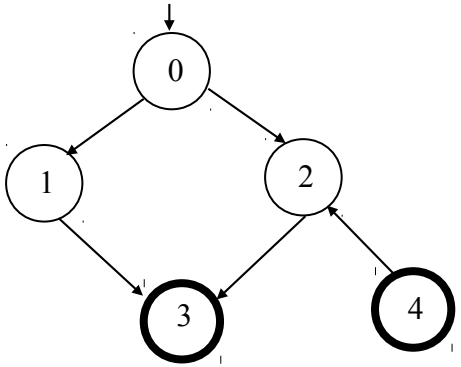


in cui il grafo ha uno ed un solo nodo iniziale ed almeno un nodo finale. Si chiede di definire il predicato `connesso` (senza argomenti), che risulta vero qualora tutti gli stati finali siano raggiungibili dallo stato iniziale (si noti che, nell'esempio, `connesso` risulta falso).

Esercizio 12

È data una base di fatti *Prolog* relativa alla specifica di un grafo aciclico diretto, come nel seguente esempio,

```
stati([0,1,2,3,4]).
iniziale(0).
finali([3,4]).
arco(0,1).
arco(0,2).
arco(1,3).
arco(2,3).
arco(4,2).
```



in cui il grafo ha uno ed un solo nodo iniziale ed almeno un nodo finale. Si chiede di definire il predicato `connesso` (senza argomenti), che risulta vero qualora tutti gli stati finali siano raggiungibili dallo stato iniziale (si noti che, nell'esempio, `connesso` risulta falso).

```
connesso :- iniziale(I), finali(F), raggiungibili(I, F).

raggiungibili(_, []).
raggiungibili(I, [H | T]) :- rag(I, H), raggiungibili(I, T).

rag(I, I).
rag(I, S) :- arco(I, S1), rag(S1, S).
```

Esercizio 13

Definire nel linguaggio *Prolog* il predicato `semisomma(L,S)`, che risulta vero quando `S` rappresenta la sommatoria degli elementi della lista `L` in posizione dispari. Per definizione, la semisomma di una lista vuota è zero.

Esercizio 13

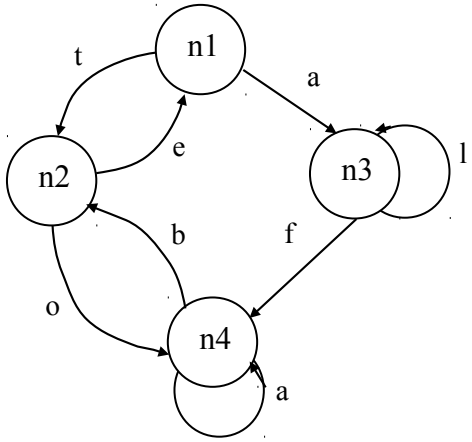
Definire nel linguaggio *Prolog* il predicato `semisomma(L,S)`, che risulta vero quando `S` rappresenta la sommatoria degli elementi della lista `L` in posizione dispari. Per definizione, la semisomma di una lista vuota è zero.

```
semisomma([],0).
semisomma([N],N).
semisomma([N1, _ | T], N) :- semisomma(T,Nt), N is N1+Nt.
```

Esercizio 14

È data una base di fatti *Prolog* relativa alla specifica di un grafo, come nel seguente esempio:

```
nodi([n1,n2,n3,n4]).
archi([arco(n1,a,n3),
       arco(n3,l,n3),
       arco(n3,f,n4),
       arco(n4,a,n4),
       arco(n4,b,n2),
       arco(n2,e,n1),
       arco(n1,t,n2),
       arco(n2,o,n4)]).
```



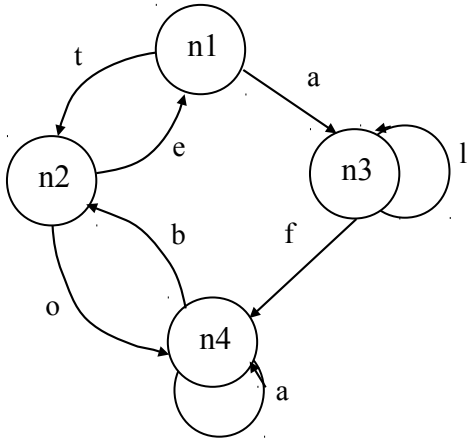
Definire la regola per il predicato **`parola(P,L,N1,N2)`**, in cui **`P`** è una lista di caratteri, di lunghezza **`L`** ≥ 0, generata da un cammino nel grafo, che parte dal nodo **`N1`** e termina al nodo **`N2`**. Ecco alcuni esempi:

P	L	N1	N2	parola(P,L,N1,N2)
[]	0	n3	n3	true
[]	1	n3	n3	false
[a]	0	n1	n3	false
[a]	1	n1	n3	true
[a,l,f,a]	4	n1	n4	true
[a,l,f,a,b,e,t,o]	8	n1	n4	true

Esercizio 14

È data una base di fatti *Prolog* relativa alla specifica di un grafo, come nel seguente esempio:

```
nodi([n1,n2,n3,n4]).
archi([arco(n1,a,n3),
       arco(n3,l,n3),
       arco(n3,f,n4),
       arco(n4,a,n4),
       arco(n4,b,n2),
       arco(n2,e,n1),
       arco(n1,t,n2),
       arco(n2,o,n4)]).
```



Definire la regola per il predicato **`parola(P,L,N1,N2)`**, in cui **`P`** è una lista di caratteri, di lunghezza **`L`** ≥ 0, generata da un cammino nel grafo, che parte dal nodo **`N1`** e termina al nodo **`N2`**.

```
parola([],0,N,N).
parola([Testa|Coda],L,N1,N2) :- L > 0,
                                archi(Archi),
                                member(arco(N1,Testa,Nc),Archi),
                                Lc is L-1,
                                parola(Coda,Lc,Nc,N2).
```

Esercizio 15

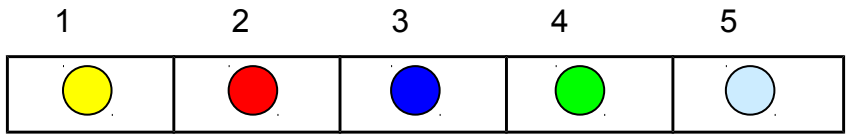
È data una scatola costituita da cinque scomparti, numerati (da sinistra a destra) 1, 2, 3, 4 e 5, e cinque palline, di colore giallo, rosso, blu, verde e azzurro. Specificare in *Prolog* il predicato

```
scatola(P1, P2, P3, P4, P5)
```

in cui `Pi` è il colore della pallina nell'i-esimo scomparto, che risulta vero se le palline sono allocate nei diversi scomparti secondo i seguenti vincoli:

- Le palline rossa e verde non sono allocate nel terzo scomparto;
- La pallina blu non è allocata ne nel primo ne nell'ultimo scomparto;
- La pallina blu è immediatamente preceduta (a sinistra) da quella rossa e immediatamente seguita (a destra) da quella verde;
- La pallina gialla è allocata a sinistra (non necessariamente nella posizione adiacente) della pallina blu;
- La pallina azzurra segue immediatamente la pallina verde.

Ecco una possibile soluzione (P1 = giallo, P2 = rosso, P3 = blu, P4 = verde, P5 = azzurro):



Esercizio 15

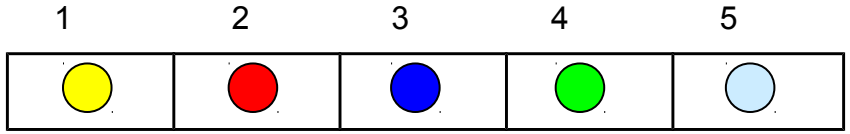
È data una scatola costituita da cinque scomparti, numerati (da sinistra a destra) 1, 2, 3, 4 e 5, e cinque palline, di colore giallo, rosso, blu, verde e azzurro. Specificare in *Prolog* il predicato

```
scatola(P1, P2, P3, P4, P5)
```

in cui `Pi` è il colore della pallina nell'i-esimo scomparto, che risulta vero se le palline sono allocate nei diversi scomparti secondo i seguenti vincoli:

- Le palline rossa e verde non sono allocate nel terzo scomparto;
- La pallina blu non è allocata ne nel primo ne nell'ultimo scomparto;
- La pallina blu è immediatamente preceduta (a sinistra) da quella rossa e immediatamente seguita (a destra) da quella verde;
- La pallina gialla è allocata a sinistra (non necessariamente nella posizione adiacente) della pallina blu;
- La pallina azzurra segue immediatamente la pallina verde.

Ecco una possibile soluzione (P1 = giallo, P2 = rosso, P3 = blu, P4 = verde, P5 = azzurro):




```
scatola(P1, P2, P3, P4, P5) :-
  Scaffali = [scaffale(P5,5),scaffale(P4,4),scaffale(P3,3),scaffale(P2,2),scaffale(P1,1)],
  member(scaffale(rosso, R), Scaffali), R \= 3,
  member(scaffale(verde, V), Scaffali), V \= 3,
  member(scaffale(blu, B), Scaffali), B \= 1, B \= 5, B is R+1, B is V-1,
  member(scaffale(giallo, G), Scaffali), G < B,
  member(scaffale(azzurro, A), Scaffali), A is V+1.
```

Esercizio 16

Assumendo di avere una base di fatti *Prolog* relativa alla specifica di una serie di famiglie, come nel seguente esempio (ogni fatto specifica il padre, la madre ed i figli di una famiglia):

```
famiglia(guido, ester, [franco, nella, elena]).
famiglia(bruno, nella, [andrea, dario, zeno]).
famiglia(franco, lucia, [giovanni, paola, letizia, sofia]).
famiglia(antonio, elena, [francesco, maria, maddalena]).
...
```

si chiede di specificare il predicato **`cugini(X,Y)`** che risulta vero qualora **`X`** ed **`Y`** siano cugini di primo grado, entrambi da parte di madre (ad esempio, **`andrea`** e **`maria`**).

Esercizio 16

Assumendo di avere una base di fatti *Prolog* relativa alla specifica di una serie di famiglie, come nel seguente esempio (ogni fatto specifica il padre, la madre ed i figli di una famiglia):

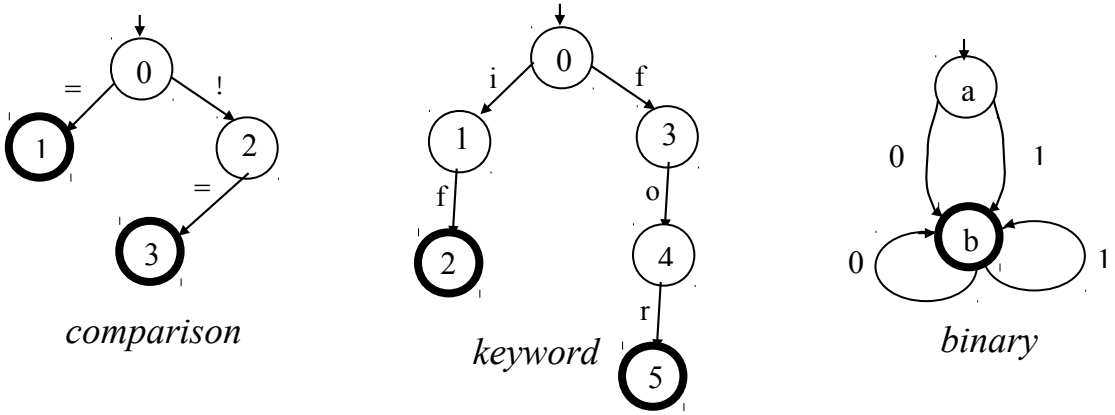
```
famiglia(guido, ester, [franco, nella, elena]).
famiglia(bruno, nella, [andrea, dario, zeno]).
famiglia(franco, lucia, [giovanni, paola, letizia, sofia]).
famiglia(antonio, elena, [francesco, maria, maddalena]).
...
```

si chiede di specificare il predicato **`cugini(X,Y)`** che risulta vero qualora **`X`** ed **`Y`** siano cugini di primo grado, entrambi da parte di madre (ad esempio, **`andrea`** e **`maria`**).

```
cugini(X,Y) :- famiglia(P1,M1,F1), famiglia(P2,M2,F2), P1 \= P2,
               member(X, F1), member(Y, F2),
               famiglia(_,_,F),
               member(M1, F), member(M2, F).
```

Esercizio 17

Si assume di avere una base di fatti *Prolog* relativa alla specifica di una serie di automi. Ogni automa è rappresentato da un nome, uno stato iniziale, un insieme di stati finali e da un insieme di transizioni. Ogni cammino dallo stato iniziale ad uno stato finale genera una parola riconosciuta dall'automa. Ecco tre automi, rispettivamente di nome *comparison*, *keyword* e *binary*:

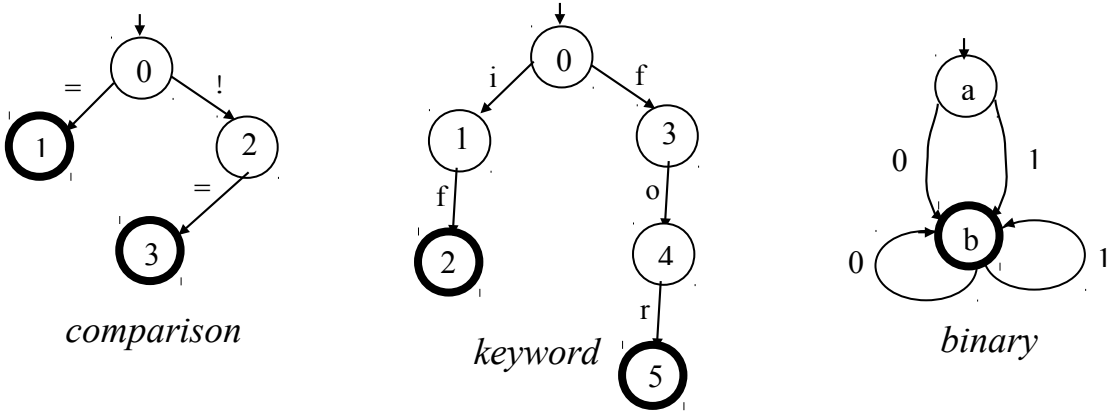


ed ecco la corrispondente base dei fatti *Prolog*:

```
automa(comparison, 0, [1,3], [tr(0,'=',1),tr(0,'!',2),tr(2,'=',3)]).
automa(keyword, 0, [2,5], [tr(0,i,1),tr(1,f,2),tr(0,f,3),tr(3,o,4),tr(4,r,5)]).
automa(binary, a, [b], [tr(a,0,b),tr(a,1,b),tr(b,0,b),tr(b,1,b)]).
```

Si chiede di specificare il predicato **`riconosce(A,P)`** che risulta vero qualora l'automa di nome **`A`** riconosca la parola **`P`**.

Esercizio 17



ed ecco la corrispondente base dei fatti *Prolog*:

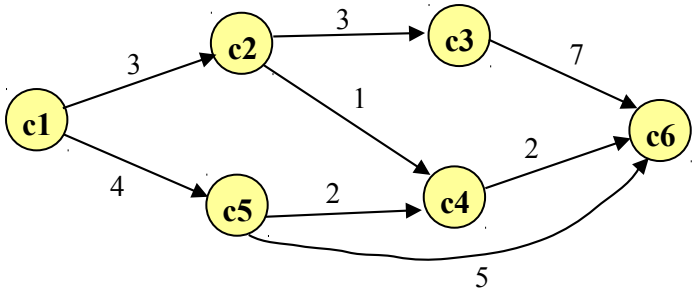
```
automa(comparison, 0, [1,3], [tr(0,'=',1),tr(0,'!',2),tr(2,'=',3)]).
automa(keyword, 0, [2,5], [tr(0,i,1),tr(1,f,2),tr(0,f,3),tr(3,o,4),tr(4,r,5)]).
automa(binary, a, [b], [tr(a,0,b),tr(a,1,b),tr(b,0,b),tr(b,1,b)]).
```

Si chiede di specificare il predicato **`riconosce(A,P)`** che risulta vero qualora l'automa di nome **`A`** riconosca la parola **`P`**.

```
riconosce(A,P) :- automa(A,I,F,T), genera(P,I,F,T).

genera([],S,F,_) :- member(S,F).
genera([Testa|Coda],S,F,T) :- member(tr(S,Testa,S2),T),
                              genera(Coda,S2,F,T).
```

Esercizio 18

E' data una base di fatti *Prolog* relativa alla specifica di una rete stradale modellata da un grafo (aciclico e diretto), in cui ogni nodo rappresenta una città, ed ogni arco rappresenta un collegamento tra due città, marcato dalla relativa distanza, come nel seguente esempio:

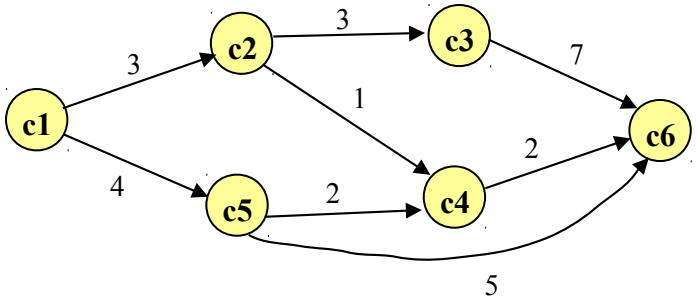



```
link(c1,3,c2).
link(c1,4,c5).
link(c2,3,c3).
link(c2,1,c4).
link(c3,7,c6).
link(c5,2,c4).
link(c5,5,c6).
link(c4,2,c6).
```

Si chiede di specificare il predicato **`percorso(C1,D12,C2)`**, che risulta vero qualora esista un cammino che collega la città **`C1`** alla città **`C2`**, di lunghezza minore o uguale a **`D12`**.

Esercizio 18

E' data una base di fatti *Prolog* relativa alla specifica di una rete stradale modellata da un grafo (aciclico e diretto), in cui ogni nodo rappresenta una città, ed ogni arco rappresenta un collegamento tra due città, marcato dalla relativa distanza, come nel seguente esempio:




```
link(c1,3,c2).
link(c1,4,c5).
link(c2,3,c3).
link(c2,1,c4).
link(c3,7,c6).
link(c5,2,c4).
link(c5,5,c6).
link(c4,2,c6).
```

Si chiede di specificare il predicato **`percorso(C1,D12,C2)`**, che risulta vero qualora esista un cammino che collega la città **`C1`** alla città **`C2`**, di lunghezza minore o uguale a **`D12`**.

```
percorso(C1,D12,C2) :- link(C1,D,C2), D =< D12.
percorso(C1,D12,C2) :- link(C1,D,C), D2 is D12 - D, percorso(C,D2,C2).
```

Esercizio 19

E' data una base di fatti *Prolog* relativa alla specifica di una grammatica BNF, in termini di assioma, simboli terminali, simboli nonterminali e produzioni, come nel seguente esempio:

$$S \rightarrow \mathbf{a}\ A \mid \mathbf{b}$$
$$A \rightarrow \mathbf{c}\ S \mid \mathbf{d}$$

⇨

```
assioma('S').
terminali([a,b,c,d]).
nonterminali(['S','A']).
produzione('S',[a,'A']).
produzione('S',[b]).
produzione('A',[c,'S']).
produzione('A',[d]).
```

Assumendo che la parte destra di ogni produzione BNF inizi sempre con un terminale (come nell'esempio), si chiede di specificare i seguenti predicati:

- **`deriva(Simboli,Frase)`**, vero qualora sia possibile derivare la frase **`Frase`** dalla lista **`Simboli`** di simboli grammaticali.
- **`corretta(Frase)`**, vero qualora **`Frase`** sia una frase del linguaggio specificato dalla BNF.

Ad esempio, **`deriva([a,'A'],[a,c,b])`** risulta vero (in quanto è possibile derivare la frase mediante i seguenti passi di derivazione: $\mathbf{a}\ A \Rightarrow \mathbf{a\ c}\ S \Rightarrow \mathbf{a\ c\ b}$). Invece, **`deriva(['S'],[a,b])`** risulta falso.

Esercizio 19

E' data una base di fatti *Prolog* relativa alla specifica di una grammatica BNF, in termini di assioma, simboli terminali, simboli nonterminali e produzioni, come nel seguente esempio:

$$S \rightarrow \mathbf{a}\ A \mid \mathbf{b}$$
$$A \rightarrow \mathbf{c}\ S \mid \mathbf{d}$$

⇨

```
assioma('S').
terminali([a,b,c,d]).
nonterminali(['S','A']).
produzione('S',[a,'A']).
produzione('S',[b]).
produzione('A',[c,'S']).
produzione('A',[d]).
```

Assumendo che la parte destra di ogni produzione BNF inizi sempre con un terminale (come nell'esempio), si chiede di specificare i seguenti predicati:

- **`deriva(Simboli,Frase)`**, vero qualora sia possibile derivare la frase **`Frase`** dalla lista **`Simboli`** di simboli grammaticali.
- **`corretta(Frase)`**, vero qualora **`Frase`** sia una frase del linguaggio specificato dalla BNF.

Ad esempio, **`deriva([a,'A'],[a,c,b])`** risulta vero (in quanto è possibile derivare la frase mediante i seguenti passi di derivazione: $\mathbf{a}\ A \Rightarrow \mathbf{a\ c}\ S \Rightarrow \mathbf{a\ c\ b}$). Invece, **`deriva(['S'],[a,b])`** risulta falso.

```
deriva([],[]).
deriva([H|T1],[H|T2]) :- deriva(T1,T2).
deriva([H|T],Frase) :- nonterminali(Nt),
                       member(H,Nt),
                       produzione(H,R),
                       append(R,T,Forma),
                       deriva(Forma, Frase).

corretta(Frase) :- assioma(A), deriva([A], Frase).
```

Esercizio 20

E' data una base di fatti *Prolog* relativa ad un albero genealogico, come nel seguente esempio:

```
capostipite(vincenzo).
genitore(vincenzo,[luisa,franco,elena]).
genitore(luisa,[andrea,dario,guido]).
genitore(franco,[giovanni,paola,letizia,sofia]).
genitore(elena,[francesco,angelo]).
...
```

in cui **`capostipite`** indica il capostipite dell'albero genealogico, mentre **`genitore`** associa ad ogni elemento dell'albero una lista (eventualmente vuota) di figli.
Si chiede di specificare il predicato **`stessa_generazione(X,Y)`**, che risulta vero qualora le persone **`X`** ed **`Y`** siano della stessa generazione (e diverse fra loro), cioè, allo stesso livello nell'albero genealogico (ad esempio, **`francesco`** e **`giovanni`**, ma non **`francesco`** e **`luisa`**).

Esercizio 20

E' data una base di fatti *Prolog* relativa ad un albero genealogico, come nel seguente esempio:

```
capostipite(vincenzo).
genitore(vincenzo,[luisa,franco,elena]).
genitore(luisa,[andrea,dario,guido]).
genitore(franco,[giovanni,paola,letizia,sofia]).
genitore(elena,[francesco,angelo]).
...
```

in cui **`capostipite`** indica il capostipite dell'albero genealogico, mentre **`genitore`** associa ad ogni elemento dell'albero una lista (eventualmente vuota) di figli.
Si chiede di specificare il predicato **`stessa_generazione(X,Y)`**, che risulta vero qualora le persone **`X`** ed **`Y`** siano della stessa generazione (e diverse fra loro), cioè, allo stesso livello nell'albero genealogico (ad esempio, **`francesco`** e **`giovanni`**, ma non **`francesco`** e **`luisa`**).

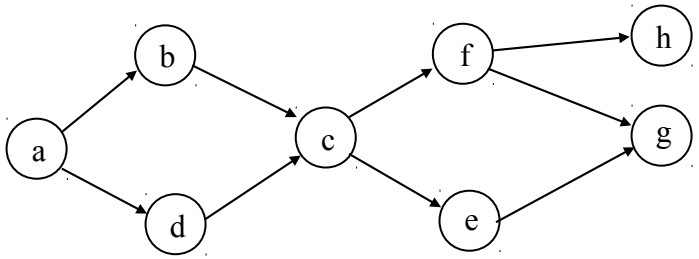
```
stessa_generazione(X, Y) :- generazione(X, G), generazione(Y, G), X \= Y.

generazione(X, 0) :- capostipite(X).
generazione(X, G) :- genitore(Y, F), member(X, F),
                     generazione(Y, Gy), G is Gy+1.
```

```
stessa_generazione(X, Y) :- genitore(_, F), member(X, F), member(Y, F), X \= Y.
stessa_generazione(X, Y) :- genitore(Zx, Fx), member(X, Fx),
                            genitore(Zy, Fy), member(Y, Fy),
                            X \= Y,
                            stessa_generazione(Zx, Zy).
```

Esercizio 21

E’ data una base di fatti *Prolog* relativa ad un grafo aciclico diretto, come nel seguente esempio,

```
arco(a,b).
arco(a,d).
arco(b,c).
arco(d,c).
arco(c,e).
arco(c,f).
arco(e,g).
arco(f,g).
arco(f,h).
```

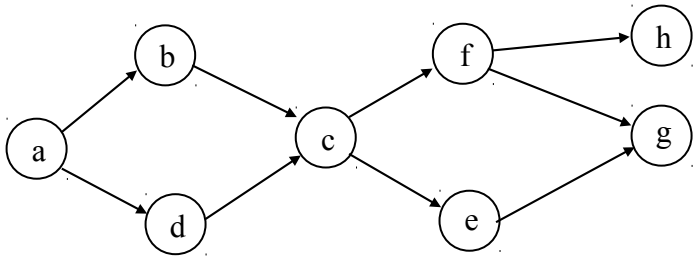


Si chiede di specificare il predicato **`path(N1,N2,P)`**, che risulta vero qualora **`P`** sia la lista di nodi generata da un cammino dal nodo **`N1`** al nodo **`N2`**, ad esempio **`path(a,f,[a,d,c,f])`**. È possibile avere anche cammini di un solo nodo, ad esempio **`path(a,a,[a])`**.

Esercizio 21

E' data una base di fatti *Prolog* relativa ad un grafo aciclico diretto, come nel seguente esempio,

```
arco(a,b).
arco(a,d).
arco(b,c).
arco(d,c).
arco(c,e).
arco(c,f).
arco(e,g).
arco(f,g).
arco(f,h).
```



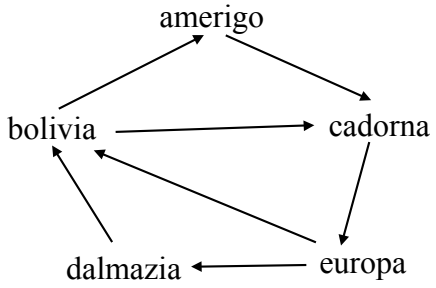
Si chiede di specificare il predicato **`path(N1,N2,P)`**, che risulta vero qualora **`P`** sia la lista di nodi generata da un cammino dal nodo **`N1`** al nodo **`N2`**, ad esempio **`path(a,f,[a,d,c,f])`**. È possibile avere anche cammini di un solo nodo, ad esempio **`path(a,a,[a])`**.

```
path(N,N,[N]).
path(N1,N2,[N1|P]) :- arco(N1,N), path(N,N2,P).
```

Esercizio 22

E’ data una base di fatti *Prolog* relativa ad una rete stradale urbana che specifica il collegamento tra piazze mediante strade a senso unico, come nel seguente esempio:

```
piazze([amerigo,bolivia,cadorna,dalmazia,europa]).

strade([strada(bolivia,amerigo),
        strada(amerigo,cadorna),
        strada(bolivia,cadorna),
        strada(cadorna,europa),
        strada(europa,dalmazia),
        strada(dalmazia,bolivia),
        strada(europa,bolivia)]).
```

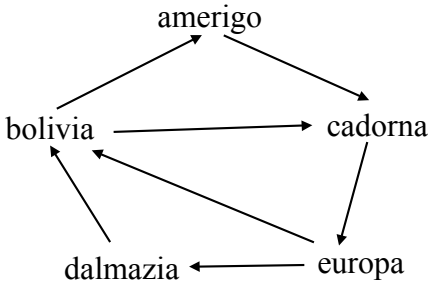


Si chiede di specificare il predicato **`percorso(Piazza)`**, che risulta vero qualora **`Piazza`** sia una piazza dalla quale sia possibile percorrere in sequenza tutte le strade (ognuna nella rispettiva direzione) una sola volta.

Esercizio 22

E' data una base di fatti *Prolog* relativa ad una rete stradale urbana che specifica il collegamento tra piazze mediante strade a senso unico, come nel seguente esempio:

```
piazze([amerigo,bolivia,cadorna,dalmazia,europa]).

strade([strada(bolivia,amerigo),
        strada(amerigo,cadorna),
        strada(bolivia,cadorna),
        strada(cadorna,europa),
        strada(europa,dalmazia),
        strada(dalmazia,bolivia),
        strada(europa,bolivia)]).
```



Si chiede di specificare il predicato **`percorso(Piazza)`**, che risulta vero qualora **`Piazza`** sia una piazza dalla quale sia possibile percorrere in sequenza tutte le strade (ognuna nella rispettiva direzione) una sola volta.

```
percorso(Piazza) :- strade(Strade), copre(Piazza,Strade).

copre(_,[]).
copre(Piazza,Strade) :-
   member(strada(Piazza,PiazzaNuova),Strade),
   rimozione(Strade,strada(Piazza,PiazzaNuova), StradeNuove),
   copre(PiazzaNuova,StradeNuove).

rimozione([H|T], H, T) :- !.
rimozione([H|T], Strada, [H|Tn]) :- rimozione(T,Strada,Tn).
```

Esercizio 23

Specificare nel linguaggio *Prolog* il predicato **`ins(N,L1,L2)`**, che risulta vero qualora **`N`** sia un numero intero, **`L1`** una lista ordinata (in modo ascendente, eventualmente vuota) di numeri interi, ed **`L2`** la lista ordinata ottenuta inserendo **`N`** in **`L1`**.

Esercizio 23

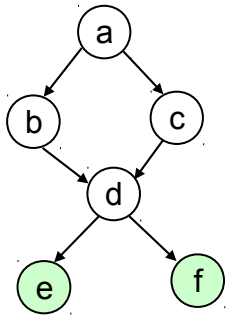
Specificare nel linguaggio *Prolog* il predicato **`ins(N,L1,L2)`**, che risulta vero qualora **`N`** sia un numero intero, **`L1`** una lista ordinata (in modo ascendente, eventualmente vuota) di numeri interi, ed **`L2`** la lista ordinata ottenuta inserendo **`N`** in **`L1`**.

```
ins(N,L1,L2) :- ordinata(L1), ordinata(L2), inserisci(N,L1,L2).

ordinata(X) :- var(X), !.
ordinata([]) :- !.
ordinata([_]) :- !.
ordinata([N1,N2|T]) :- N1 =< N2, ordinata([N2|T]).

inserisci(N,[],[N]) :- !.
inserisci(N,[H|T],[N,H|T]) :- N =< H, !.
inserisci(N,[H|T],[H|Z]) :- inserisci(N,T,Z).
```

Esercizio 24

La base dei fatti di un programma *Prolog* descrive un grafo aciciclo e diretto, come nel seguente esempio:

```
arco(a,b).
arco(a,c).
arco(b,d).
arco(c,d).
arco(d,e).
arco(d,f).
```

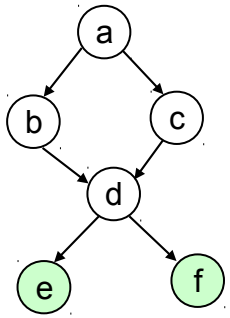


Si chiede di specificare il predicato **`foglie(F)`**, che risulta vero qualora **`F`** sia una lista costituita unicamente da nodi foglia del grafo (nell'esempio, i nodi foglia sono colorati in verde).

Esercizio 24

La base dei fatti di un programma *Prolog* descrive un grafo aciciclo e diretto, come nel seguente esempio:

```
arco(a,b).
arco(a,c).
arco(b,d).
arco(c,d).
arco(d,e).
arco(d,f).
```



Si chiede di specificare il predicato **`foglie(F)`**, che risulta vero qualora **`F`** sia una lista costituita unicamente da nodi foglia del grafo (nell'esempio, i nodi foglia sono colorati in verde).

```
foglie([]).
foglie([H|T]) :- arco(_,H), \+(arco(H,_)), foglie(T).
```

Esercizio 25

Specificare nel linguaggio *Prolog* il predicato **`positivi(N,P)`**, che risulta vero qualora **`P`** sia la sottolista dei numeri positivi (maggiori di zero) della lista di numeri **`N`**.

Esercizio 25

Specificare nel linguaggio *Prolog* il predicato **`positivi(N,P)`**, che risulta vero qualora **`P`** sia la sottolista dei numeri positivi (maggiori di zero) della lista di numeri **`N`**.

```
positivi([],[]).
positivi([N|Tn],[N|Tp]) :- N > 0, positivi(Tn,Tp).
positivi([N|Tn],P) :- N =< 0, positivi(Tn,P).
```

Esercizio 26

Specificare nel linguaggio *Prolog* il predicato **`media(L,M)`**, che risulta vero qualora **`M`** sia la media dei numeri nella lista **`L`**.

Esercizio 26

Specificare nel linguaggio *Prolog* il predicato **`media(L,M)`**, che risulta vero qualora **`M`** sia la media dei numeri nella lista **`L`**.

```
media(L,M) :- length(L,N), average(L,N,M).

average([I], N, M) :- M is I/N, !.
average([H|T], N, M) :- M1 is H/N,
                        average(T, N, Mt),
                        M is M1 + Mt.
```

Esercizio 27

È dato un fatto *Prolog* relativo ai tasselli di un domino, come nel seguente esempio:

```
tasselli([t(3,4),t(5,3),t(4,1),t(1,5)]).
```

Si chiede di specificare il predicato **`domino(X)`**, che risulta vero qualora **`X`** sia una lista composta da tutti i tasselli ordinati secondo le regole del domino. Nel nostro esempio avremmo:

```
?-  domino(X).
X = [t(4, 1), t(1, 5), t(5, 3), t(3, 4)] ;
X = [t(3, 4), t(4, 1), t(1, 5), t(5, 3)] ;
X = [t(1, 5), t(5, 3), t(3, 4), t(4, 1)] ;
X = [t(5, 3), t(3, 4), t(4, 1), t(1, 5)] ;
false.
```

Esercizio 27

È dato un fatto *Prolog* relativo ai tasselli di un domino, come nel seguente esempio:

```
tasselli([t(3,4),t(5,3),t(4,1),t(1,5)]).
```

Si chiede di specificare il predicato **`domino(X)`**, che risulta vero qualora **`X`** sia una lista composta da tutti i tasselli ordinati secondo le regole del domino. Nel nostro esempio avremmo:

```
?-  domino(X).
X = [t(4, 1), t(1, 5), t(5, 3), t(3, 4)] ;
X = [t(3, 4), t(4, 1), t(1, 5), t(5, 3)] ;
X = [t(1, 5), t(5, 3), t(3, 4), t(4, 1)] ;
X = [t(5, 3), t(3, 4), t(4, 1), t(1, 5)] ;
false.
```

```
domino(X) :- tasselli(L), member(T,L), componibile(X,L,[T]).

componibile(X,L,X) :- length(L,N), length(X,N).
componibile(X,L,A) :- member(T,L),
                      \+(member(T,A)),
                      T = t(_,N2),
                      A = [t(N2,_)|_],
                      componibile(X,L,[T|A]).
```

Esercizio 28

È dato un fatto *Prolog* che specifica una lista di numeri interi, come nel seguente esempio:

```
numeri([-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10]).
```

Si chiede di specificare il predicato **`scomponi(A,B,C,D)`**, che risulta vero qualora **`A`** e **`B`** siano i coefficienti di una equazione di secondo grado $x^2 + Ax + B = 0$ che possa essere espressa come prodotto $(x+C)(x+D) = 0$, in cui **`C`** e **`D`** appartengono alla lista argomento del fatto **`numeri`**. Si ricorda che, ai fini della scomposizione, **`A`** e **`B`** devono corrispondere rispettivamente alla somma ed al prodotto di **`C`** e **`D`**. Ad esempio, $x^2 + 3x - 4 = 0$ può essere espressa come $(x-1)(x+4) = 0$, quindi:

```
?- scomponi(3,-4,C,D).
C = -1,
D = 4.
```

Si richiede anche che la regola sia specificata in modo tale che l'interprete dia un'unica soluzione (nel nostro esempio, la seconda soluzione (simmetrica, ma che non deve essere fornita) sarebbe **`C = 4, D = -1`**).

Esercizio 28

È dato un fatto *Prolog* che specifica una lista di numeri interi, come nel seguente esempio:

```
numeri([-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10]).
```

Si chiede di specificare il predicato **`scomponi(A,B,C,D)`**, che risulta vero qualora **`A`** e **`B`** siano i coefficienti di una equazione di secondo grado $x^2 + Ax + B = 0$ che possa essere espressa come prodotto $(x+C)(x+D) = 0$, in cui **`C`** e **`D`** appartengono alla lista argomento del fatto **`numeri`**. Si ricorda che, ai fini della scomposizione, **`A`** e **`B`** devono corrispondere rispettivamente alla somma ed al prodotto di **`C`** e **`D`**. Ad esempio, $x^2 + 3x - 4 = 0$ può essere espressa come $(x-1)(x+4) = 0$, quindi:

```
?- scomponi(3,-4,C,D).
C = -1,
D = 4.
```

Si richiede anche che la regola sia specificata in modo tale che l'interprete dia un'unica soluzione (nel nostro esempio, la seconda soluzione (simmetrica, ma che non deve essere fornita) sarebbe **`C = 4, D = -1`**).

```
scomponi(A,B,C,D) :- numeri(N),
                     member(C,N), member(D,N),
                     A is C+D,
                     B is C*D, !.
```

Esercizio 29

È data la base di fatti *Prolog* relativa ad una mappa di tracciati per corse ad ostacoli. La mappa è costituita da un grafo aciclico diretto, i cui nodi rappresentano bandierine e i cui archi rappresentano tratti di percorso. Ogni arco è marcato dal numero di ostacoli coinvolti in quel tratto. Ad esempio, il tratto dalla bandierina **`a`** alla bandierina **`b`**, che coinvolge **`10`** ostacoli, viene rappresentato in *Prolog* dal seguente fatto:

```
tratto(a,b,10).
```

Si chiede di specificare il predicato **`corsa(X,Y,N)`**, che risulta vero qualora esista un tracciato (sequenza di tratti) che parte dalla bandierina **`X`**, termina alla bandierina **`Y`** e coinvolge **`N`** ostacoli in totale.

Esercizio 29

È data la base di fatti *Prolog* relativa ad una mappa di tracciati per corse ad ostacoli. La mappa è costituita da un grafo aciclico diretto, i cui nodi rappresentano bandierine e i cui archi rappresentano tratti di percorso. Ogni arco è marcato dal numero di ostacoli coinvolti in quel tratto. Ad esempio, il tratto dalla bandierina **`a`** alla bandierina **`b`**, che coinvolge **`10`** ostacoli, viene rappresentato in *Prolog* dal seguente fatto:

```
tratto(a,b,10).
```

Si chiede di specificare il predicato **`corsa(X,Y,N)`**, che risulta vero qualora esista un tracciato (sequenza di tratti) che parte dalla bandierina **`X`**, termina alla bandierina **`Y`** e coinvolge **`N`** ostacoli in totale.

```
corsa(X,Y,N) :- tratto(X,Y,N).
corsa(X,Y,N) :- tratto(X,Z,M), corsa(Z,Y,K), N is M+K.
```

Esercizio 30

È data la base di fatti *Prolog* che rappresenta un albero. Ogni fatto specifica un frammento di albero, cioè il collegamento di un nodo padre con i suoi figli, come nel seguente esempio (in cui **`p`** è il padre e **`f1`**, **`f2`**, **`f3`** i figli):

```
frammento(p,[f1, f2, f3]).
```

Si chiede di specificare il predicato **`equinodi(N1, N2)`**, che risulta vero qualora **`N1`** ed **`N2`** siano due nodi (diversi fra loro) posizionati allo stesso livello nell'albero.

Esercizio 30

È data la base di fatti *Prolog* che rappresenta un albero. Ogni fatto specifica un frammento di albero, cioè il collegamento di un nodo padre con i suoi figli, come nel seguente esempio (in cui **`p`** è il padre e **`f1`**, **`f2`**, **`f3`** i figli):

```
frammento(p,[f1, f2, f3]).
```

Si chiede di specificare il predicato **`equinodi(N1, N2)`**, che risulta vero qualora **`N1`** ed **`N2`** siano due nodi (diversi fra loro) posizionati allo stesso livello nell'albero.

```
equinodi(N1,N2) :- frammento(_,F), member(N1,F), member(N2,F), N1 \= N2.
equinodi(N1,N2) :- frammento(P1,F1), frammento(P2,F2),
                   member(N1,F1), member(N2,F2),
                   equinodi(P1,P2).
```

Esercizio 31

È data una base di fatti *Prolog* che specifica la grammatica EBNF di un linguaggio **L** in termini di assioma, simboli nonterminali, simboli terminali e produzioni. L'unico operatore esteso della grammatica è l'opzionalità, espressa dal funtore `opt`, con possibilità di innestamento. Ecco un esempio:

$S \rightarrow \mathbf{a}\ A\ [\mathbf{b}] \mid \mathbf{c}$
$A \rightarrow \mathbf{b}\ [B\ \mathbf{c}\ [\mathbf{d}]] \mid \mathbf{e}$
$B \rightarrow \mathbf{f}\ S \mid \mathbf{a}$

```
assioma('S').
terminali([a,b,c,d,e,f]).
nonterminali(['S','A','B']).
produzione('S',[a,'A',opt([b])]).
produzione('S',[c]).
produzione('A',[b,opt(['B',c,opt([d])])]).
produzione('A',[e]).
produzione('B',[f,'S']).
produzione('B',[a]).
```

Assumendo che la parte destra di ogni produzione inizi sempre con un simbolo terminale, si chiede di specificare in *Prolog* il predicato `frase(F)`, che risulta vero se e solo se `F` è una frase del linguaggio `L`.

Esercizio 31

È data una base di fatti *Prolog* che specifica la grammatica EBNF di un linguaggio **L** in termini di assioma, simboli nonterminali, simboli terminali e produzioni. L'unico operatore esteso della grammatica è l'opzionalità, espressa dal funtore `opt`, con possibilità di innestamento. Ecco un esempio:

$S \rightarrow \mathbf{a}\ A\ [\mathbf{b}] \mid \mathbf{c}$
$A \rightarrow \mathbf{b}\ [B\ \mathbf{c}\ [\mathbf{d}]] \mid \mathbf{e}$
$B \rightarrow \mathbf{f}\ S \mid \mathbf{a}$

```
assioma('S').
terminali([a,b,c,d,e,f]).
nonterminali(['S','A','B']).
produzione('S',[a,'A',opt([b])]).
produzione('S',[c]).
produzione('A',[b,opt(['B',c,opt([d])])]).
produzione('A',[e]).
produzione('B',[f,'S']).
produzione('B',[a]).
```

Assumendo che la parte destra di ogni produzione inizi sempre con un simbolo terminale, si chiede di specificare in *Prolog* il predicato `frase(F)`, che risulta vero se e solo se `F` è una frase del linguaggio `L`.

```
frase(F) :- assioma(S), deriva([S],F).

deriva([],[]).
deriva([H|T1],[H|T2]) :- deriva(T1,T2).
deriva([H|T],F) :- nonterminali(Nt),
                   member(H,Nt),
                   produzione(H,R),
                   append(R,T,L),
                   deriva(L, F).
deriva([opt(O)|T],F) :- append(O,T,L), deriva(L,F).
deriva([opt(_)|T],F) :- deriva(T,F).
```

Esercizio 32

Specificare nel linguaggio *Prolog* il predicato `specchio(L, S)`, che risulta vero quando `S` è la lista speculare di `L`, come nel seguente esempio:

```
> specchio([1, 2, 3, [a, b, c], [], x, y, z], S).
S = [z, y, x, [], [c, b, a], 3, 2, 1].
```

(Si consiglia di definire il predicato ausiliario `lista(X)`, che risulta vero quando `X` è una lista).

Esercizio 32

Specificare nel linguaggio *Prolog* il predicato `specchio(L, S)`, che risulta vero quando `S` è la lista speculare di `L`, come nel seguente esempio:

```
> specchio([1, 2, 3, [a, b, c], [], x, y, z], S).
S = [z, y, x, [], [c, b, a], 3, 2, 1].
```

(Si consiglia di definire il predicato ausiliario `lista(X)`, che risulta vero quando `X` è una lista).

```
lista([]).
lista([_|_]).

specchio([],[]).

specchio([H|T],R) :- lista(H),
                     specchio(H,RH),
                     specchio(T,RT),
                     append(RT,[RH],R).

specchio([H|T],R) :- \+(lista(H)),
                     specchio(T,RT),
                     append(RT,[H],R).
```

Esercizio 33

È data una base di fatti *Prolog* che specifica la grammatica BNF di un linguaggio in termini di simboli nonterminali (di cui il primo è l'assioma), simboli terminali e produzioni. Ecco un esempio:

$S \rightarrow \mathbf{a}\ A\ \mathbf{b}$
$A \rightarrow A\ \mathbf{c}\ B$
$B \rightarrow \mathbf{c}\ A\ \mathbf{c} \mid \mathbf{a}\ \mathbf{c}\ B$
$C \rightarrow \mathbf{c}\ A \mid \mathbf{a}$

```
terminali([a,b,c]).
nonterminali(['S','A','B','C']).
produzioni([prod('S',[a,'A',b]),
            prod('A',['A',c,'B']),
            prod('B',[c,'A',c]),
            prod('B',[a,c,'B']),
            prod('C',[c,'A']),
            prod('C',[a])]).
```

Si chiede di specificare in *Prolog* il predicato `ricorsiva(N)`, che risulta vero se e solo se la grammatica include una produzione (direttamente) ricorsiva relativa al nonterminale `N`. Ad esempio:

```
?- ricorsiva(X).
X = 'A' ;
X = 'B' ;
false.
```

Esercizio 33

È data una base di fatti *Prolog* che specifica la grammatica BNF di un linguaggio in termini di simboli nonterminali (di cui il primo è l'assioma), simboli terminali e produzioni. Ecco un esempio:

$S \rightarrow \mathbf{a}\ A\ \mathbf{b}$
$A \rightarrow A\ \mathbf{c}\ B$
$B \rightarrow \mathbf{c}\ A\ \mathbf{c} \mid \mathbf{a}\ \mathbf{c}\ B$
$C \rightarrow \mathbf{c}\ A \mid \mathbf{a}$

```
terminali([a,b,c]).
nonterminali(['S','A','B','C']).
produzioni([prod('S',[a,'A',b]),
            prod('A',['A',c,'B']),
            prod('B',[c,'A',c]),
            prod('B',[a,c,'B']),
            prod('C',[c,'A']),
            prod('C',[a])]).
```

Si chiede di specificare in *Prolog* il predicato `ricorsiva(N)`, che risulta vero se e solo se la grammatica include una produzione (direttamente) ricorsiva relativa al nonterminale `N`. Ad esempio:

```
?- ricorsiva(X).
X = 'A' ;
X = 'B' ;
false.
```

```
ricorsiva(N) :- nonterminali(NT),
                produzioni(PR),
                member(N, NT),
                member(prod(N, RHS), PR),
                member(N, RHS).
```

Esercizio 34

È data una base di fatti *Prolog* che specifica un grafo aciclico diretto, come nel seguente esempio:

```
nodi([0, 4, 5, 8, 9, 10]).

archi([arco(0, 2, 10),
       arco(0, 3, 4),
       arco(10, 3, 5),
       arco(4, 6, 5),
       arco(5, -1, 8),
       arco(5, 4, 9)]).
```

Si assume implicitamente che il nodo iniziale sia il primo della lista di nodi. Ogni nodo è identificato da un numero intero. Ogni arco tra due nodi è marcato da un numero intero (ad esempio, `arco(10, 3, 5)` indica un arco dal nodo 10 al nodo 5, marcato dal numero 3). Si chiede di specificare il predicato `bilanciato(X)`, che risulta vero se e solo se esiste un cammino nel grafo che, partendo dal nodo iniziale, raggiunge il nodo `X` in modo tale che la somma dei numeri che marcano gli archi di tale cammino coincida con `X` (nel caso di cammino nullo, tale somma vale zero per definizione).

Esercizio 34

È data una base di fatti *Prolog* che specifica un grafo aciclico diretto, come nel seguente esempio:

```
nodi([0, 4, 5, 8, 9, 10]).

archi([arco(0, 2, 10),
       arco(0, 3, 4),
       arco(10, 3, 5),
       arco(4, 6, 5),
       arco(5, -1, 8),
       arco(5, 4, 9)]).
```

Si assume implicitamente che il nodo iniziale sia il primo della lista di nodi. Ogni nodo è identificato da un numero intero. Ogni arco tra due nodi è marcato da un numero intero (ad esempio, `arco(10, 3, 5)` indica un arco dal nodo 10 al nodo 5, marcato dal numero 3). Si chiede di specificare il predicato `bilanciato(X)`, che risulta vero se e solo se esiste un cammino nel grafo che, partendo dal nodo iniziale, raggiunge il nodo `X` in modo tale che la somma dei numeri che marcano gli archi di tale cammino coincida con `X` (nel caso di cammino nullo, tale somma vale zero per definizione).

```
bilanciato(S) :- nodi([I|_]), cammino(I, 0, S).

cammino(N, N, N).
cammino(S1, N, S2) :- archi(A),
                      member(arco(S1, N12, S), A),
                      M is N + N12,
                      cammino(S, M, S2).
```

Esercizio 35

È data una base di fatti *Prolog* che specifica la grammatica BNF di un linguaggio in termini di assioma, simboli terminali, simboli nonterminali e produzioni, come nel seguente esempio:

$S \rightarrow \mathbf{x}\ A\ \mathbf{y}$
$A \rightarrow \mathbf{y}\ S\ \mathbf{z} \mid C\ \mathbf{y}$
$B \rightarrow A\ \mathbf{w} \mid \mathbf{x}\ B$
$C \rightarrow B\ \mathbf{z} \mid \mathbf{x}$

```
assioma('S').
terminale(x).
terminale(y).
terminale(z).
terminale(w).
nonterminale('S').
nonterminale('A').
nonterminale('B').
nonterminale('C').
produzione('S',[x,'A',y]).
produzione('A',[y,'S',z]).
produzione('A',['C',y]).
produzione('B',['A',w]).
produzione('B',[x,'B']).
produzione('C',['B',z]).
produzione('C',[x]).
```

Si chiede di specificare in *Prolog* il predicato `ricorsione(N)`, che risulta vero se e solo se esiste una produzione del nonterminale `N` che è (direttamente o indirettamente) ricorsiva, cioè quando è possibile derivare da `N` una forma sentenziale che inizia con il simbolo `N`. Nel nostro esempio si avrebbe:

```
?- ricorsione(N).
N = 'A' ;
N = 'B' ;
N = 'C'.
```

Esercizio 35

È data una base di fatti *Prolog* che specifica la grammatica BNF di un linguaggio in termini di assioma, simboli terminali, simboli nonterminali e produzioni, come nel seguente esempio:

$S \rightarrow \mathbf{x}\ A\ \mathbf{y}$
$A \rightarrow \mathbf{y}\ S\ \mathbf{z} \mid C\ \mathbf{y}$
$B \rightarrow A\ \mathbf{w} \mid \mathbf{x}\ B$
$C \rightarrow B\ \mathbf{z} \mid \mathbf{x}$

```
assioma('S').
terminale(x).
terminale(y).
terminale(z).
terminale(w).
nonterminale('S').
nonterminale('A').
nonterminale('B').
nonterminale('C').
produzione('S',[x,'A',y]).
produzione('A',[y,'S',z]).
produzione('A',['C',y]).
produzione('B',['A',w]).
produzione('B',[x,'B']).
produzione('C',['B',z]).
produzione('C',[x]).
```

Si chiede di specificare in *Prolog* il predicato `ricorsione(N)`, che risulta vero se e solo se esiste una produzione del nonterminale `N` che è (direttamente o indirettamente) ricorsiva, cioè quando è possibile derivare da `N` una forma sentenziale che inizia con il simbolo `N`. Nel nostro esempio si avrebbe:

```
?- ricorsione(N).
N = 'A' ;
N = 'B' ;
N = 'C'.
```

```
ricorsione(N) :- nonterminale(N), deriva(N, N).
deriva(N, H) :- produzione(N, [H|_]).
deriva(N, H) :- produzione(N, [X|_]), nonterminale(X),
                deriva(X, H), !.
```

Esercizio 36

Definire nel linguaggio Prolog il predicato `cat(L, C)`, in cui `L` è una lista di liste e `C` una lista, il quale risulta vero qualora `C` sia la concatenazione di tutte le liste di `L`.

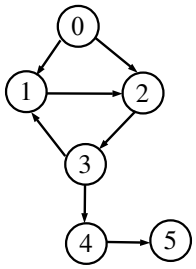
Esercizio 36

Definire nel linguaggio Prolog il predicato `cat(L, C)`, in cui `L` è una lista di liste e `C` una lista, il quale risulta vero qualora `C` sia la concatenazione di tutte le liste di `L`

```
cat([], []).
cat([H|T], C) :- cat(T, Z), append(H,Z,C).
```

Esercizio 37

È data una base di fatti *Prolog* che definisce un grafo orientato ciclico, come nel seguente esempio:

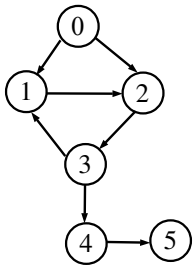



```
nodi([0,1,2,3,4,5]).
archi([arco(0,1),
       arco(0,2),
       arco(1,2),
       arco(2,3),
       arco(3,1),
       arco(3,4),
       arco(4,5)]).
```

Si chiede di specificare il predicato `raggiunge(X,Y)`, vero se e solo esiste un cammino che parte dal nodo `X` e termina nel nodo `Y`. (Nota: Essendo il grafo ciclico, è necessario evitare loop nella ricerca ...).

Esercizio 37

È data una base di fatti *Prolog* che definisce un grafo orientato ciclico, come nel seguente esempio:




```
nodi([0,1,2,3,4,5]).
archi([arco(0,1),
       arco(0,2),
       arco(1,2),
       arco(2,3),
       arco(3,1),
       arco(3,4),
       arco(4,5)]).
```

Si chiede di specificare il predicato `raggiunge(X,Y)`, vero se e solo esiste un cammino che parte dal nodo `X` e termina nel nodo `Y`. (Nota: Essendo il grafo ciclico, è necessario evitare loop nella ricerca ...).

```
raggiunge(X,Y) :- rag(X,Y,[]).

rag(X,X,_).
rag(X,Y,_) :- archi(A), member(arco(X,Y),A).
rag(X,Y,R) :- archi(A), member(arco(X,Z),A),
              \+(member(Z,R)),
              rag(Z,Y,[Z|R]).
```

Esercizio 38

Specificare nel linguaggio *Prolog* il predicato `bysum(X)`, in cui `X` è una lista di interi, il quale risulta vero quando ogni numero in `X` è la somma di due numeri qualunque in `X`.

Esercizio 38

Specificare nel linguaggio *Prolog* il predicato `bysum(X)`, in cui `X` è una lista di interi, il quale risulta vero quando ogni numero in `X` è la somma di due numeri qualunque in `X`.

```
bysum(Ns) :- bysum2(Ns,Ns).

bysum2([],_) :- !.
bysum2([X|Y],Ns) :- bs(X,Ns), bysum2(Y,Ns).

bs(N,Ns) :- member(N1,Ns), member(N2,Ns), N is N1 + N2, !.
```

Esercizio 39

Specificare nel linguaggio *Prolog* il predicato `posizioni(N1,N2,N3,N4,N5,P1,P2,P3,P4,P5)` i cui argomenti sono numeri interi (diversi fra loro), il quale risulta vero quando ogni `Pi`, `i` $\in$ [1..5], rappresenta l'i-esimo numero in un ordinamento ascendente dei numeri `N1...N5`, come nel seguente esempio:

```
?- posizioni(34,12,1,78,0,P1,P2,P3,P4,P5).
P1 = 0,
P2 = 1,
P3 = 12,
P4 = 34,
P5 = 78
```

Esercizio 39

Specificare nel linguaggio *Prolog* il predicato `posizioni(N1,N2,N3,N4,N5,P1,P2,P3,P4,P5)` i cui argomenti sono numeri interi (diversi fra loro), il quale risulta vero quando ogni `Pi`, `i` $\in$ [1..5], rappresenta l'i-esimo numero in un ordinamento ascendente dei numeri `N1...N5`, come nel seguente esempio:

```
?- posizioni(34,12,1,78,0,P1,P2,P3,P4,P5).
P1 = 0,
P2 = 1,
P3 = 12,
P4 = 34,
P5 = 78
```

```
posizioni(N1, N2, N3, N4, N5, P1, P2, P3, P4, P5) :-
  Lista = [P1,P2,P3,P4,P5],
  member(N1, Lista),
  member(N2, Lista),
  member(N3, Lista),
  member(N4, Lista),
  member(N5, Lista), P1 < P2, P2 < P3, P3 < P4, P4 < P5.
```

Esercizio 40

Specificare nel linguaggio *Prolog* il predicato `esiste(X)`, in cui `X` rappresenta una lista di numeri, il quale risulta vero quando esiste un numero `N` in `X` uguale alla somma dei due numeri che precedono `N` in `X`.

Esercizio 40

Specificare nel linguaggio *Prolog* il predicato `esiste(X)`, in cui `X` rappresenta una lista di numeri, il quale risulta vero quando esiste un numero `N` in `X` uguale alla somma dei due numeri che precedono `N` in `X`.

```
esiste([X,Y,Z|_]) :- Z is X + Y, !.
esiste([_|Q]) :- esiste(Q).
```

Esercizio 41

Specificare nel linguaggio *Prolog* il predicato `perogni(X)`, in cui `X` rappresenta una lista di numeri, il quale risulta vero quando la lista ha meno di tre numeri o ogni numero `N` in `X` è uguale alla somma dei due numeri che precedono `N` in `X`.

Esercizio 41

Specificare nel linguaggio *Prolog* il predicato `perogni(X)`, in cui `X` rappresenta una lista di numeri, il quale risulta vero quando la lista ha meno di tre numeri o ogni numero `N` in `X` è uguale alla somma dei due numeri che precedono `N` in `X`.

```
perogni([]) :- !.
perogni([_]) :- !.
perogni([_,_]) :- !.
perogni([X,Y,Z|Q]) :- Z is X + Y, perogni([Y,Z|Q]).
```

Esercizio 42

Specificare nel linguaggio *Prolog* il predicato `uguali(X,Y)`, in cui `X` ed `Y` denotano una lista (istanziata) di interi che rappresenta un multi-insieme (cioè, un insieme con possibili duplicati, come ad esempio `[1,3,2,4,3,5,2,3,6,4]`), il quale risulta vero quando `X` ed `Y` includono gli stessi numeri ed ogni numero compare lo stesso numero di volte (eventualmente in posizioni diverse) nelle due liste.

Esercizio 42

Specificare nel linguaggio *Prolog* il predicato `ugualiX,Y`, in cui `X` ed `Y` denotano una lista (istanziata) di interi che rappresenta un multi-insieme (cioè, un insieme con possibili duplicati, come ad esempio `[1,3,2,4,3,5,2,3,6,4]`), il quale risulta vero quando `X` ed `Y` includono gli stessi numeri ed ogni numero compare lo stesso numero di volte (eventualmente in posizioni diverse) nelle due liste.

```
ugualiX,Y) :- lengthX,Nx), lengthY,Ny), Nx == Ny, isomorfiX,Y), !.

isomorfi[],[]) :- !.
isomorfi[H|T],Y) :- memberH,Y), removeH,Y,R), isomorfiT,R).

removeX,[X|T], T) :- !.
removeX, [H|T], [H|Tr]) :- removeX, T, Tr).
```

Esercizio 43

Specificare in *Prolog* il predicato `quadrupla(X)`, in cui `X` è una lista (istanziata) di interi, che risulta vero quando `X` contiene una sequenza strettamente ordinata di quattro numeri, in cui il quarto numero è la somma dei primi due numeri (ad esempio `[... 8,10,12,18,...]`).

Esercizio 43

Specificare in *Prolog* il predicato `quadrupla(X)`, in cui `X` è una lista (istanziata) di interi, che risulta vero quando `X` contiene una sequenza strettamente ordinata di quattro numeri, in cui il quarto numero è la somma dei primi due numeri (ad esempio `[... 8,10,12,18,...]`).

```
quadrupla([N1,N2,N3,N4|_]) :- N1 < N2, N2 < N3, N3 < N4, N4 is N1 + N2, !.
quadrupla([_|Coda]) :- quadrupla(Coda).
```

Esercizio 44

Specificare in *Prolog* il predicato `ordinata(L)`, in cui `L` è una lista (istanziata) di interi, che risulta vero quando `L` o contiene meno di tre numeri o ogni numero dal terzo in poi corrisponde alla somma dei due numeri precedenti, come nel seguente esempio: `[5,7,12,19,31,50]`.

Esercizio 44

Specificare in *Prolog* il predicato `ordinata(L)`, in cui `L` è una lista (istanziata) di interi, che risulta vero quando `L` o contiene meno di tre numeri o ogni numero dal terzo in poi corrisponde alla somma dei due numeri precedenti, come nel seguente esempio: `[5,7,12,19,31,50]`.

```
ordinata([]).
ordinata([_]).
ordinata([_,_]).
ordinata([X, Y, Z | T]) :- Z is X + Y, ordinata([Y, Z | T]).
```

Esercizio 45

Specificare in *Prolog* il predicato `quadrati(N,Q)`, in cui `N` è un intero maggiore o uguale a zero, mentre `Q` è la lista dei quadrati dei numeri interi da `1` a `N`. Si assume che `N` sia istanziato.

Esercizio 45

Specificare in *Prolog* il predicato `quadrati(N,Q)`, in cui `N` è un intero maggiore o uguale a zero, mentre `Q` è la lista dei quadrati dei numeri interi da `1` a `N`. Si assume che `N` sia istanziato.

```
quadrati(0,[]) :- !.
quadrati(N,Q) :- N1 is N-1,
                 NN is N*N,
                 quadrati(N1,Q1),
                 append(Q1, [NN], Q).
```

Esercizio 46

Specificare in *Prolog* il predicato `singletons(L,S)`, in cui `L` è una lista (anche vuota), mentre `S` è la lista di liste (singleton) dei singoli elementi di `L`.

Esercizio 46

Specificare in *Prolog* il predicato `singletons(L,S)`, in cui `L` è una lista (anche vuota), mentre `S` è la lista di liste (singleton) dei singoli elementi di `L`.

```
singletons([],[]) :- !.
singletons([H|T],[[H]|ST]) :- singletons(T,ST).
```

Esercizio 47

Specificare in *Prolog* il predicato `swapairs(L,S)`, in cui `L` è una lista (anche vuota) di un numero pari di elementi, mentre `S` è la lista di elementi scambiati a coppie, come nel seguente esempio:

```
swapairs([a,b,c,d,e,f], [b,a,d,c,f,e])
```

Esercizio 47

Specificare in *Prolog* il predicato `swapairs(L,S)`, in cui `L` è una lista (anche vuota) di un numero pari di elementi, mentre `S` è la lista di elementi scambiati a coppie, come nel seguente esempio:

```
swapairs([a,b,c,d,e,f], [b,a,d,c,f,e])
```

```
swapairs([],[]) :- !.
swapairs([X,Y|T],[Y,X|SP]) :- swapairs(T,SP).
```

Esercizio 48

Specificare in *Prolog* il predicato `sumpairs(L,S)`, in cui `L` è una lista (anche vuota) di numeri, mentre `S` è la lista di somme di coppie, come nel seguente esempio:

```
sumpairs([1,2,3,4,8,12], [3,7,20])
```

Esercizio 48

Specificare in *Prolog* il predicato `sumpairs(L,S)`, in cui `L` è una lista (anche vuota) di numeri, mentre `S` è la lista di somme di coppie, come nel seguente esempio:

```
sumpairs([1,2,3,4,8,12], [3,7,20])
```

```
sumpairs([],[]) :- !.
sumpairs([X],[X]) :- !.
sumpairs([X,Y|T],[Z|SP]) :- Z is X+Y, sumpairs(T,SP).
```