

## Fratello - Genitore - Nonno - Avo



Program × +

```
1 genitore(mario,dario).
2 genitore(mario,gino).
3 genitore(gino,pino).
4 genitore(gino,sandro).
5 genitore(sandro,luca).
6 genitore(luca,mario).
7
8 fratello(X,Y):-
9     genitore(Z,X),
10    genitore(Z,Y).
11
12 nonno(X,Y):-
13     genitore(X,Z),
14     genitore(Z,Y).
15 avo(X,Y):-
16     genitore(X,Y).
17
18 avo(X,Y):-
19     genitore(X,Z),
20     avo(Z,Y).
21
```



## Archi e Percorsi

```
Program x +
1  /* edge(S,E) */
2
3  edge(a,b).
4  edge(b,c).
5  edge(c,d).
6  edge(a,e).
7  edge(e,f).
8  edge(f,k).
9  edge(f,c).
10
11
12  path(a,m).
13
14  %PB
15  path(X,Y) :-
16      edge(X,Y).
17
18
19  %PI
20  path(X,Y) :-
21      edge(X,Z),
22      path(Z,Y).
```

## MEMBER

```
1  /* appartiene(X,L) */
2  /* caso1 X appartiene ad H --> Sostituiamo H con X --> L corrisponderà a [X|T]
3  ---> Sostituiamo T con _ ---> L = [X|_] */
4  appartiene(X,[X|_]).
5
6  appartiene(X,[_|T]):-
7      appartiene(X,T).
```

## OPERATORE SOMMA (SWISH)

  Program ✕ +

```
1
2 :- op(300,yfx,somma).
3
4 somma(t(X,Y,Z),S):-
5     S is X+Y+Z.
```





## CONCATENA

```
10 /* PB: */
11 concatena([],A,A).
12
13 /* INDUZ STRUTT */
14 concatena([H|T],B,[H|L]):-
15     concatena(T,B,L).
```

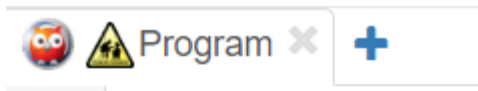
## RIVOLTARE LISTA

```
1 /* INDUZIONE STRUTTURALE rivoltata( L, RL ) */
2 |
3 rivoltata([],[]).
4
5
6 rivoltata([H|T], RL):-
7     rivoltata(T,RT),
8     append(RT,[H],RL).
9
```

## PERMUTAZIONE

```
Program    
1 append([],A,A).
2 append((H|T),B,[H|L]):-
3     append(T,B,L).
4
5 rivoltata([],[]).
6 rivoltata([H|T],RL):-
7     append(RT,[H],RL),
8     rivoltata(T,RT).
9
10 appartiene(X,[X|_]).
11 appartiene(X,[_|T]):-
12     appartiene(X,T).
13
14 permutazione([],[]).
15 permutazione([H|T],B):-
16     permutazione(T,PT1_2),
17     appartiene(H,B),
18     subtract(H,B,PT1_2).
19
20 subtract(_,[],[]).
21 subtract(H,[H|R],R).
22 subtract(H,[A|R],[A|R2]):-
23     subtract(H,R1,R2).
24
```

## LUNGHEZZA



```
1 lung([], 0).
2
3 lung([_|T], N) :-
4     lung(T,M),
5     M >= 0,
6     N is M + 1.
7
```

## CONTATORE ELEMENTO

```
11 /* numero_di_elementi(Lista, Elemento, Numero) */
12
13 numero_di_el([], _, 0).
14 numero_di_el([El|T], El, N) :-
15     numero_di_el(T, El, M),
16     N is M + 1.
17 numero_di_el([X|T], El, M):-
18     X \= El,
19     numero_di_el(T,El,M).
20
```

## FALLIMENTO

```
30 %fail
31 fallimento_di_g(A):-
32     g(A),!,fail.
33 fallimento_di_g(_).
```

## MYNOT:

ciò che è dentro è vero, tutto il resto è falso.

```

36 %negation as failure
37 mynot(Predicato):-
38     Predicato,!,fail.
39 mynot(_).

```

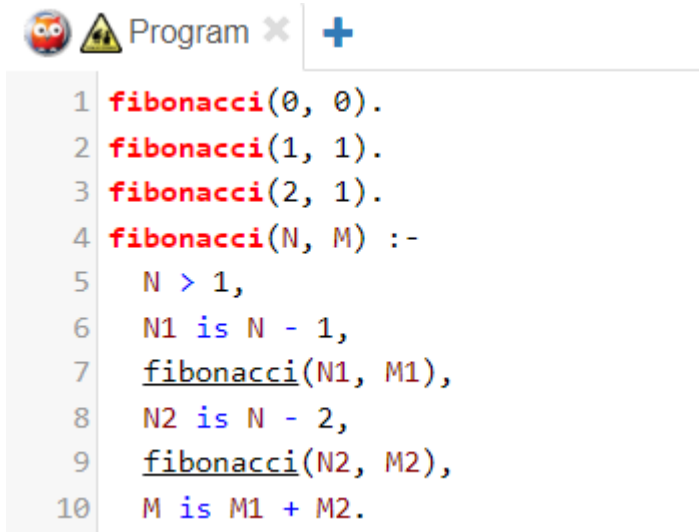
## Num\_elementi

```

43 %num_elementi(X,L,N).
44
45 num_elementi(_,[],0).
46 num_elementi(X,[X|T],N):-
47     !,
48     num_elementi(X,T,N1),
49     N is N1 + 1.
50 num_elementi(X,[_|T],N):-
51     num_elementi(X,T,N).

```

## Predicato Prolog per la funzione Fibonacci (NO DYNAMIC)



```

1 fibonacci(0, 0).
2 fibonacci(1, 1).
3 fibonacci(2, 1).
4 fibonacci(N, M) :-
5     N > 1,
6     N1 is N - 1,
7     fibonacci(N1, M1),
8     N2 is N - 2,
9     fibonacci(N2, M2),
10    M is M1 + M2.

```

## Predicato Prolog per la funzione Fibonacci (DYNAMIC)

Composizione del predicato senza assert.

Program x +

```
1 :- dynamic f/2.
2 fibonacci(0, 0).
3 fibonacci(1, 1).
4 fibonacci(2, 1).
5 fibonacci(N, M) :-
6     write(in),nl,
7     N1 is N - 1,
8     fibonacci(N1, M1),
9     N2 is N - 2,
10    fibonacci(N2, M2),
11    M is M1 + M2.
```

## Fibonacci (dynamic [assert] )

Program x +

```
1 :- dynamic fibonacci/2.
2 fibonacci(0, 0).
3 fibonacci(1, 1).
4 fibonacci(2, 1).
5 fibonacci(N, M) :-
6     write(in(N,M)),nl,
7     N1 is N - 1,
8     N2 is N - 2,
9     N1 > 0,
10    N2 > 0,
11    fibonacci(N1, M1),
12    fibonacci(N2, M2),
13    M is M1 + M2,
14    asserta(fibonacci(N,M)),
15    !.
```

## STRUTTURA ALBOREA / PREDICATO leaf & node

Program ✕ +

```
1 leaf(t(R,[]), R). % ritorna vero se R sono uguali e se R ha [] figli (solo le foglie non hanno figli)
2
3 leaf(t(_,Child),L):-
4     member(C,Child),
5     leaf(C,L).
6
7
8 node(t(R,_),R).
9
10 node(t(_,Child),L):- % ritorna ver se R sono uguali e se R ha _(qualsiasi cosa) figli
11     member(C,Child),
12     node(C,L).|
```

## OPERATORE “HA” (possessione) E “COSA”

```
8 :- op(100,yfx,di).
9 :- op(300,yfx,ha).
10
11 mario ha macchina di dario.
12 giovanni ha panino.
13 elena ha panino di giovanni.
14 giacomo ha borse di pelle di daino.
15
16 /* ? Chi ha cosa */
```

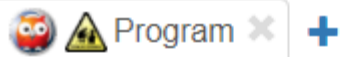


## LA TORRE DI HANOI

  Program ✕   Program ✕ +

```
1 hanoi([], _, _).
2 hanoi(h([X|A],B,C),h(A,[X|B],C)):-
3     ordinata([X|A]),
4     ordinata([X|B]),
5     ordinata(C).
6
7 ordinata([]).
8 ordinata([_]).
9 ordinata([H1,H2|T]):-
10     H1>H2,
11     ordinata([H2|T]).
12 |
```

## **“BAGOF”**



```
1 :- dynamic appoggio/1.
2
3 n(11).
4 n(2).
5 n(4).
6 n(5).
7 n(8).
8
9 appoggio([]).
10
11 numeri(L):-
12     n(Num),
13
14     appoggio(L),
15     write(L),nl,
16     append(L,[Num],LN),
17     retract(appoggio(L)),
18     assert(appoggio(LN)),
19
20     write(Num),nl,
21     fail.
22
23
24 numeri(L):-
25     appoggio(L),
26     retract(appoggio(L)),
27     assert(appoggio([])),
28     write(fine),nl.
```

## PREDICATO BAGOF E SETOF


  Program ✕

```
1  
2 n(11).  
3 n(2).  
4 n(4).  
5 n(5).  
6 n(8).  
7 n(4).  
8 n(8).  
9
```

 `bagof(N,n(N),L).`

`L = [11, 2, 4, 5, 8, 4, 8]`

?- `bagof(N,n(N),L).`

 `setof(N,n(N),L).`

`L = [2, 4, 5, 8, 11]`


?- `setof(N,n(N),L).`

- **bagof** raccoglie tutti gli elementi che soddisfano l'obiettivo, inclusi i duplicati.
- **setof** raccoglie solo elementi univoci che soddisfano l'obiettivo e li ordina.

Restituisce per ogni lettera il corrispondente numerico


---

```
1
2 n(11,a).
3 n(2,m).
4 n(4,k).
5 n(5,g).
6 n(8,f).
7 n(4,f).
8 n(8,d).
9
10
11 l(L):-
12     n(_,L).
13 num(L):-
14     n(L,_).
```

 bagof(N,L^n(N,L),Lista).

**Lista** = [11, 2, 4, 5, 8, 4, 8]

?- bagof(N,L^n(N,L),**Lista**).

 L=k, bagof(N,L^n(N,L),Lista).

**L** = k,

**Lista** = [4]

?- L=k, bagof(N,L^n(N,L),**Lista**).

# PROBLEMA DELLE 8 REGINE



```
1 controlloriga([A,B],[A,C]).
2
3 controllocolonna([A,B],[C,A]).
4
5 controllodiagonale([A,B],[C,D]):-
6     X is A-C,
7     Y is B-D,
8     X \= Y,
9     X \= -Y.
10
11 controllocoppia([A]).
12
13 controllocoppia([A,B|T]):-
14     controlloriga(A,B),
15     controllocolonna(A,B),
16     controllodiagonale(A,B),
17     controllocoppia([A|T]).
18
19 controllosoluzione([A]).
20
21 controllosoluzione([H|T]):-
22     \+controllocoppia([H|T]),
23     controllosoluzione([T]).
```

## VISITA DFS

```
3
4  /* DFS */
5  edge(a,b).
6  edge(b,c).
7  edge(c,d).
8  edge(a,e).
9  edge(e,f).
10 edge(f,k).
11 edge(f,c).
12
13 path(a,m).
14
15 path(X,Y,[X,Y]) :-
16     edge(X,Y).
17
18 path(X,Y,[X|P_Z_Y]):-
19     edge(X,Z),
20     path(Z,Y,P_Z_Y).
21
```

## VISITA BFS

### 1° parte ( controllo delle frontiere )

```
23 /* BFS */
24
25 /* Pf(F,PF). */
26
27 prossimafrontiera([], []).
28
29 prossimafrontiera([X|R], F):-
30     setof(Z,edge(X,Z),RZ), %Lista di nodi raggiungibili
31     prossimafrontiera(R,FF),
32     append(RX,RF,F).|
```

## 2° parte ( Verifica del path attraverso le frontiere )


```
23 /* BFS */
24
25 /* Pf(F,PF). */
26
27 prossimafrontiera([], []).
28
29 prossimafrontiera([[X,PX]|R], F):-
30     setof([Z|[X|PX]],edge(X,Z),RZ), %Lista di nodi raggiungibili
31     prossimafrontiera(R,FF),
32     append(RX,RF,F).
33
34 opf(F,FR,Y):-
35     prossimafrontiera(F,FR),
36     member(Y,FR).
37 opf(F,FR,Y):-
38     prossimafrontiera(F,FRZ),
39     opf(FRZ,FR,Y).
40
41 path(X,Y,P):-
42     prossimafrontiera([X|R],F),
43     member(Y,F),
44     reverse(F,P).
45     opf([X],FF,Y).
46
```

# IL PROBLEMA DEL PERCORSO DEL CAVALLO

```
1 controllo_unicita(_, []).
2
3 controllo_unicita(A/B, [A/C|T]):-
4     B \= C,
5     controllo_unicita(A/B,T).
6
7 controllo_unicita(A/B, [_/B|T]):-
8     A \= C,
9     controllo_unicita(A/B,T).
10
11 controllo_unicita(A/B, [_/_|T]):-
12     A \= C,
13     B \= D,
14     controllo_unicita(A/B,T).
15
16 controllo_unicita_totale([_]).
17 controllo_unicita_totale([M|T]):-
18     controllo_unicita(M,T),
19     controllo_unicita_totale(T).
20
21 mossa_valida(A/B,C/D):-
22     abs(A-C, 1),
23     abs(B-D, 2).
24
25 mossa_valida(A/B,C/D):-
26     abs(A-C, 2),
27     abs(B-D, 1).
28
29 controllo_mosse([_]).
30
31 controllo_mosse([M1, M2|T]):-
32     mossa_valida(M1,M2),
33     controllo_mosse([M2|T]).
34
35 controllo_soluzione(L):-
36     controllo_unicita_totale(L),
37     controllo_mosse(L).
38
39
```



## GRAMMATICA IN PROLOG

 Program  

```
1 % data una grammatica dire a quali stringhe appartengono al linguaggio
2
3 % data la stringa baa dire se appartiene al linguaggio
4
5 'A'(L,R1):-
6     'B'(L,R),
7     'C'(R,R1).
8 'A'(L,R2):-
9     'C'(L,R),
10    'B'(R,R1),
11    'B'(R1,R2).
12
13 /*Scrivi la seguenti query (?- listing.) e sostituisci le righe 15 e 16
14 'B'-->'a'.
15 'A'-->'b'.
16 */
17
18 %Sostituisci con :
19
20 'B'([H|T],T):-|
21     H = 'a'.
22 'C'([H|T],T):-
23     H = 'b'.
```

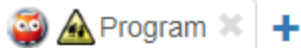
## SPLIT

```
Program x +
1 split([],[],[]).
2
3 split(L,LS,LD):-
4     append(LS,LD,L),
5     LS=[_|_],
6     LD=[_|_].
7
8 operazione(X,Y,X+Y).
9 operazione(X,Y,X-Y).
10 operazione(X,Y,X*Y).
11 operazione(X,Y,X/Y):-
12     Y \= 0.
13
14 termine([X],X).
15
16 termine(L, T):-
17     split(L,LS,LD),
18     termine(LS,TS),
19     termine(LD,TD),
20     operazione(TS,TD,T).
21
22 equazione(L):-
23     split(L,LS,LD),
24     termine(LS,TS),
25     termine(LD,TD),
26     TS := TD.
```

## OPERAZIONI SU GRAMMATICHE CF

```
Program x +
1 a(K)-->b(K),c,d,{n(K)}.
2
3 'S'(K) --> [up],'S'(M),{K is M+1}.
4
5 'S'(K) --> [down],'S'(M),{K is M-1}.
6
7 'S'(1) --> [up].
8
9 'S'(-1) --> [down].
10
11
```

## SIMULAZIONE ESAME TEGOLE E TETTO



```
1 rettangolo(Base,0,[]).
2
3 rettangolo(Base,Altezza,[X|Rett]):-
4     Altezza > 0,
5     length(X,Base),
6     AltezzaMeno1 is Altezza - 1,
7     rettangolo(Base,AltezzaMeno1,Rett).
8 pannello(a).
9 pannello(b).
10 pannello(c).
11 pannello(d).
12
13 lista_completa([]).
14 lista_completa([A|R]):-
15     nonvar(A),
16     lista_completa(R).
17
18 lista_pannelli(lungh,lista):-
19     length(lista,lungh),
20     lista_pannelli(lista).
21
22 /* lista_pannelli(lista):-
23     lista_completa(lista).
24
25 lista_pannelli(lista):-
26     pannello(X),
27     write(X),nl,
28     member(X,lista),
29     lista_pannelli(lista). */
30
31 lista_pannelli([]).
32
33 |
34 lista_pannelli([X|lista]):-
35     pannello(X),
36     lista_pannelli(lista).
37
```