# Eserciziario PROLOG


## Prof Aldo Franco Dragoni

Università Politecnica delle Marche


### Esercizio 1.

Guardare e capire l'esecuzione del seguente programma, lanciando il goal
`?-frase`

```
% es1-0.pl
word(article,a).
word(article,every).
word(noun,criminal).
word(noun,'big Mac').
word(verb,eats).
word(verb,likes).

frase :-
        word(article,Word1),
        word(noun,Word2),
        word(verb,Word3),
        word(article,Word4),
        word(noun,Word5),
        write(Word1),tab(1),
        write(Word2),tab(1),
        write(Word3),tab(1),
        write(Word4),tab(1),
        write(Word5),tab(1),nl,
        fail.1-0
```

### Esercizio 2.

Si consideri il seguente programma Prolog:

```
salute(giorgio).
salute(anna).
salute(marco).
salute(maria).
salute(alberto).
ama(giorgio,elisa).
ama(massimo,elisa).
ama(marco,maria).
ama(elisa,giorgio).
ama(anna,giorgio).
ama(maria,alberto).
ama(alberto,maria).
```

dove `salute/1` viene interpretato in "essere in salute" e `ama/2` in "X ama Y".
(a) Aggiungere al programma opportune regole che definiscano un predicato `felice/1` che venga interpretato in "essere felice", dove X è felice se è in salute e ama qualcuno da cui è ricambiato.% % X è felice se è in salute e ama qualcuno da cui è ricambiato.
(b) Aggiungere al programma opportune regole che definiscano il predicato `felici/2` in modo che questa venga interpretato in "X e Y sono felici" (dove due persone si dicono felici se entrambe sono in salute e ciascuna di esse ama l'altra).

**Soluzione 2.**

```
% X è felice se è in salute e ama qualcuno da cui è ricambiato.
felice(X):- salute(X),
      ama(X,Y),
      ama(Y,X).
% due persone si dicono felici se entrambe sono in salute e ciascuna di esse
ama l'altra
felici(X,Y):- salute(X),
      salute(Y),
      ama(X,Y),
      ama(Y,X).
```

## Esercizio 3.

Scrivere due predicati per calcolare l'intersezione e l'unione di due "insiemi" liste.

```
?- intersezione([1,3,5,2,4],[6,1,2],L).
L = [1, 2] ;
false.

?- unione([1,3,5,2,4],[6,1,2],L).
L = [3, 5, 4, 6, 1, 2] ;
false.
```

## Esercizio 4.

Scrivere un programma che sostituisca un intero scelto con un atomo letterale scelto lasciando inalterato l'ordine.

```
?- sost([1,3,1,2,1,8],1,a,L).
L = [a, 3, a, 2, a, 8] ;
false.
```

**Esercizio 5.**

Individuare il significato del seguente predicato

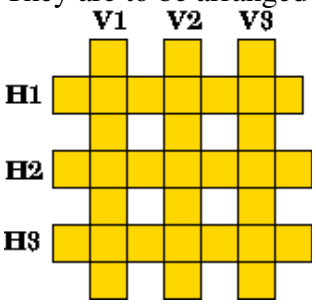
```
genera([_|C],[X|C1]):-
      member(X,[1,x,2]),
      genera(C,C1).
genera([],[]).
```

utilizzando ad esempio i seguenti test

```
?- genera([1,2],L).
?- genera([1,2,3],L).
```

**Esercizio 6.**

Here are six English words:
*abalone, abandon, anagram, connect, elegant, enhance.*
They are to be arranged in a crossword puzzle like fashion in the grid given below.




The following knowledge base represents a lexicon containing these words.

```
word(abalone,a,b,a,l,o,n,e).
word(abandon,a,b,a,n,d,o,n).
word(enhance,e,n,h,a,n,c,e).
word(anagram,a,n,a,g,r,a,m).
word(connect,c,o,n,n,e,c,t).
word(elegant,e,l,e,g,a,n,t).
```

**Esercizio 7. Crivello di Eratostene**

Il procedimento è il seguente: si scrivono tutti i naturali a partire da 2 fino ad *n* in un elenco detto setaccio. Poi si cancellano (setacciano) tutti i multipli del primo numero del setaccio (escluso lui stesso). Si prosegue così fino ad arrivare in fondo. I numeri che restano sono i numeri primi minori od uguali a n. È come se si utilizzassero dei setacci a maglie via via più larghe: il primo lascia passare solo i multipli di 2, il secondo solo i multipli di 3, e così via.

```
?- primi(100,L,5).

   2   3   5   7  11S
  13  17  19  23  29
  31  37  41  43  47
  53  59  61  67  71
  73  79  83  89  97

L=[2, 3, 5, 7, 11, 13, 17, 19, 23|...] ;
false.
```

**Esercizio 8.**

Scrivere un programma PROLOG che, dopo aver chiesto e ottenuto in input una lista `L`, chieda (fino a quando non viene immesso `fine`) un atomo `A`. Se l'atomo appartiene alla lista, ne trovi la posizione nella lista stessa e stampi: "`l'atomo A si trova nella posizione N`" (la posizione del primo elemento corrisponda a 1). Se l'atomo non appartiene alla lista stampi : "`l'atomo A non appartiene alla lista`".

**Esercizio 9.**

Un albero binario non vuoto `alb(X,Sin,Des)` è ordinato da sinistra a destra (e viene detto "dizionario binario") se:
1) Tutti i nodi nel sottoalbero di sinistra, `Sin`, sono minori di `X`;
2) Tutti i nodi nel sottoalbero di destra, `Des` , sono maggiori di `X`;
3) entrambi i sottoalberi sono loro stessi ordinati.
Scrivere un programma PROLOG che legga da tastiera dei numeri interi positivi (fino a quando non viene battuto `stop`) e costruisca un dizionario binario.
I numeri inseriti non sono duplicati.
Scrivere inoltre un predicato che, dato un dizionario binario, ne stampi i nodi (non vuoti) in ordine crescente.

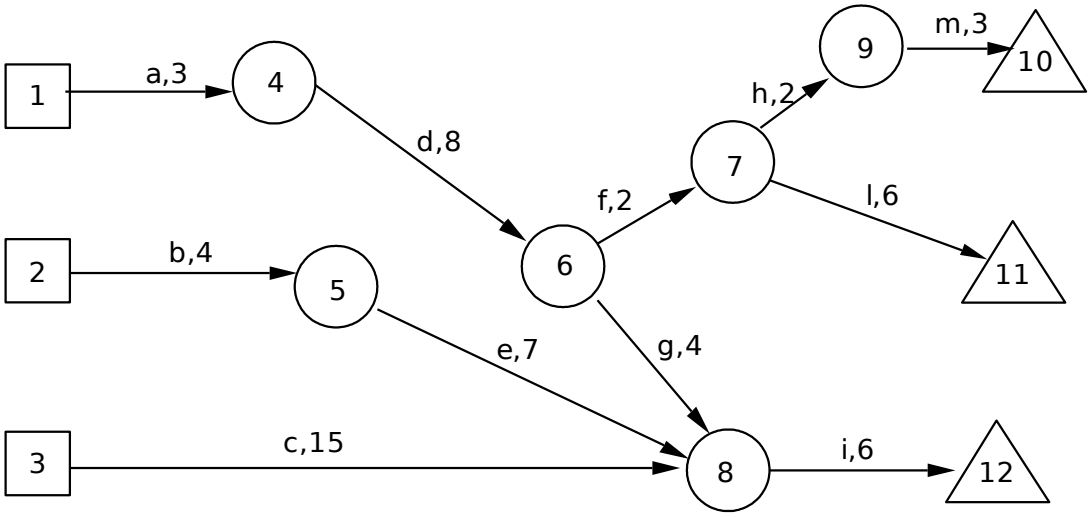
**Esercizio 10.**

Scrivere un predicato PROLOG `trova_elemento` che, dopo aver chiesto e ottenuto in input da tastiera una lista, chiede (fino a quando non viene immesso `fine`) un numero intero `N` e stampi l'elemento che si trova nella `N`-ma posizione nella lista secondo lo schema-esempio riportato sotto. Venga controllato che `N` non sia né maggiore della lunghezza della lista, né minore di `1`. Il primo elemento della lista corrisponda a `N`=`1`

```
?-trova_elemento.
dai la lista [1,8,7,5,4].
dai N 3.
l'elemento n. 3 e' 7
dai N 9.
errore su N, riprova
dai N 2.
l'elemento n. 2 e' 8
dai N fine.
yes
```

**Esercizio 11.**

In riferimento alla figura, scrivere un programma PROLOG che determini il minimo percorso che unisce uno dei nodi identificati da un quadrato con uno dei nodi identificati con un triangolo. Gli archi hanno una lunghezza pari al numeri posto accanto a ciascuna label. Il percorso venga restituito come una lista ordinata delle label degli archi utili.




**Esercizio 12.**

Scrivere un predicato PROLOG `piu_lunga/3` che, preso al primo argomento una lista di liste di interi, restituisca al secondo quella che contiene più elementi, ordinata in modo crescente; restituisca poi al terzo argomento la lunghezza della lista restituita. Nel caso di liste di ugual lunghezza ne venga scelta una qualunque. Per l'ordinamento si utilizzi il metodo del quick-sort.

```
?- piu_lunga([[8,7,3],[9,2],[1,7,8,3]],L,LUNG).
L = [1,3,7,8]
LUNG = 4
```

**Esercizio 13.**

Se si vuol calcolare la somma di due numeri naturali maggiori del massimo numero rappresentabile sul sistema si potrebbe pensare di rappresentare ogni numero come una lista delle sue cifre. Scrivere un programma PROLOG `run/0` che legga da tastiera i due numeri naturali (scritti come atomi PROLOG) e ne stampi la somma, come da esempio sotto riportato. Il codice ASCII di `0` è `48`.

```
?- run.
dai il primo numero '670192'.
dai il secondo numero '9539718'.
la somma di 670192 e di 9539718 e' 10209910
```

**Esercizio 14.**

Scrivere un predicato PROLOG `run/0` che:

- permetta l'introduzione da tastiera di una successione di numeri naturali terminata da uno `0` <$r_1, r_2, \ldots . r_n, 0$>
- stampi il numero di volte che compare l'ultimo elemento (escluso lo `0`) se questo e' minore di `10`, o il suo primo antecedente minore di `10` se questo è maggiore di `10`; qualora non se ne trovi nessuno minore di `10` se ne stampi opportuna dicitura.

```
?- run.
8.
1.
8.
14.
```

```
0.
ci sono 2 elementi uguali a 8
yes

?- run.
14.
19.
0.
nessun elemento < 0
yes
```

**Esercizio 15.**

Un albero si può rappresentare come lista (di liste) (`[a,b,c]`), oppure come struttura con un apposito funtore (`alb(a,alb(b,nil,nil),alb(c,nil,nil))`). Scrivere un predicato PROLOG `trasla/2` che restituisca al secondo argomento l'albero binario espresso come lista preso al primo argomento, nel suo equivalente espresso come struttura con funtore `alb` e viceversa. Prima di procedere alla traslazione si verifichi che l'albero istanziato passato sia effettivamente un albero binario.

```
?- trasla([a,b,[c,d,e]],A).
A = alb(a,alb(b,nil,nil),alb(c,alb(d,nil,nil),alb(c,nil,nil)))
yes
?- trasla(A,alb(a,alb(b,nil,nil),alb(c,alb(d,nil,nil),alb(c,nil,nil)))).
A = [a,b,[c,d,e]]
yes
```

**Esercizio 16.**

Scrivere un predicato PROLOG `trova_posizione/0` che, dopo aver chiesto e ottenuto una lista in input, chiede fino a quando non viene battuto `fine`, un atomo e ne stampi la posizione nella lista. Se l'atomo non viene trovato venga stampata opportuna dicitura (vedi esempio). La posizione del primo elemento della lista è 1.

```
?-  trova_posizione.
dai la lista: [1,a,3,b].
dai l'elemento da trovare: a.
l'elemento a si trova nella posizione n. 2
dai l'elemento da trovare: 4.
l'elemento non é presente nella lista
dai l'elemento da trovare: fine.
```

**Esercizio 17.**

Scrivere un predicato PROLOG `lettere/1`, che prende in ingresso un atomo e stampi in ordine alfabetico tutte e sole le lettere minuscole che lo compongono (tutti gli altri caratteri vengano trascurati) con la relativa frequenza. Le lettere minuscole hanno un codeice ASCII compreso tra 97 e 122.

```
?- lettere('Tutti i miei amici').
 lettera    presenza
      a           1
      c           1
      e           1
      i           6
      m           2
      t           2
      u           1
```

**Esercizio 18.**

Si supponga di avere nel Data Base PROLOG fatti del tipo `a(X,Y)`, dove `X` e `Y` sono atomi (non necessariamente numerici). Si scriva un predicato PROLOG `somme(S1,S2)` che istanzi `S1` ed `S2` rispettivamente alla somma dei primi e dei secondi argomenti numerici dei fatti del tipo sopra citato contenuti nel Data Base.

**Esercizio 19.**

Sia data una lista non ordinata di parole (atomi) che possono essere ripetute. Scrivere un programma PROLOG che restituisca una lista contenente ogni parola che compare nella lista con la relativa frequenza di comparizione, cominciando da quella con frequenza massima fino a quella con frequenza minima. (Si utilizzi il quick-sort).

**Esercizio 20.**

Sia data una lista non ordinata di parole (atomi) che possono essere ripetute. Scrivere un programma PROLOG che restituisca una lista contenente ogni parola che compare nella lista con la relativa frequenza di comparizione, cominciando da quella con frequenza massima fino a quella con frequenza minima. (Si utilizzi il quick-sort).

**Esercizio 21.**

Scrivere un predicato PROLOG `ord(LI,A,B,LO)` che, avendo `LI` istanziata ad una lista di interi ed `A` e `B` a due interi (con `A < B`), istanzi `LO` a tutti gli interi della lista `LI` esterni a `[A , B]` ordinati in ordine crescente. Per l'ordinamento si utilizzi il metodo "merge-sort".

**Esercizio 22.**

Scrivere un programma PROLOG `medie(X,N)`, che legga da tastiera numeri interi e stampi di volta in volta la media. Il programma deve scartare tutti i numeri maggiori di `N` e deve terminare quando viene immesso lo `0`. Prima di ogni lettura deve essere segnalato il numero degli interi accettati fino a quel momento e la loro media. Si utilizzi uno schema come nell'esempio che segue:

```
Sono stati immessi 4 numeri la cui media è 16.3.
Dai il prossimo numero:
```

Alla fine il programma deve unificare successivamente la variabile `X` con ciascuno dei numeri immessi e accettati, nell'ordine in cui gli stessi sono stati forniti dall'utente.

**Esercizio 23.**

Scrivere un predicato PROLOG `piu_piccolo(L,V)`, in cui `L` è una lista di termini del tipo `a(I)`, con `I` intero positivo, che istanzi `V` al termine `a(I)` con il più piccolo `I`.
Scrivere inoltre un predicato `ordina(L,L1)`, in cui `L` é una lista come quella sopradescritta e `L1` la lista `L` ordinata in maniera decrescente rispetto ad `I`. (Nell'ordinamento si utilizzi il metodo del quick-sort)

```
?- piu_piccolo([a(5),a(1),a(2)],V).
A = a(1)

?- ordina([a(5),a(1),a(2)],L1).
L1 = [a(5),a(2),a(1)]
```

**Esercizio 24.**

Scrivere un programma PROLOG che data una lista `L` di liste di numeri interi ed un numero intero `N>0` restituisca la lista ordinata in senso crescente composta da tutti gli `N`-mi elementi di ciascuna delle liste componenti `L`. Qualora `N` sia maggiore della lunghezza di una delle liste, venga trascurato il contributo della lista in questione. Per l'ordinamento si utilizzi il metodo del quicksort.

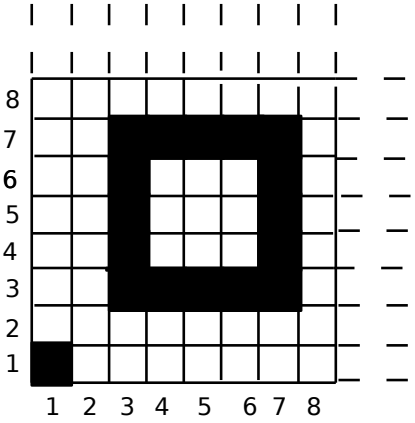
```
?-  n_mi_el([2,7,8],[1,2],[4,5,6],3,L).
L=[6,8]
```

**Esercizio 25.**

Si supponga di rappresentare una matrice rettangolare di pixel, come quella presentata in figura,

utilizzando fatti del tipo `on(X,Y)` il cui significato inteso è che il pixel di coordinate `(X,Y)` è acceso. Scrivere i seguenti predicati:

- `punto(X,Y)` che verifichi e/o restituisca le coordinate `(X,Y)` di un pixel acceso isolato
- `segmento(X1,Y1,X2,Y2)` che verifichi e/o restituisca le coppie di coordinate `(X1,Y1)`, `(X2,Y2)` di due punti, `P1` e `P2`, estremi di un segmento orizzontale, verticale o diagonale, costituito da tutti pixel accesi
- `rettangolo(X1,Y1,X2,Y2,X3,Y3,X4,Y4)` che verifichi che le quattro coppie di coordinate corrispondano a quattro punti costituenti, ordinatamente, i vertici di un rettangolo; i lati del rettangolo siano costituiti da tutti pixel accesi e siano paralleli ai lati della matrice di pixel.




```
?- punto(X,Y).
X=1
Y=1
?- segmento(X1,Y1,X2,Y2).
X1=3
Y1=3
X2=7
Y2=3
?- rettangolo(3,3,7,3,7,7,3,7).
yes
```

**Esercizio 26.**

Si considerino i tre simboli del totocalcio `1`, `x` e `2`. Sia *<partita>* un tipo di lista contenente o un simbolo solo (giocata fissa) o due (doppia) o tutt'e tre (tripla). Sia *<sistema>* una lista di (eventualmente 13) liste del tipo *<partita>*. Si definisca un predicato `sviluppo(`*<sistema>*`,MaxX,Max1,Max2)` dove `MaxX`, `Max1` e `Max2` sono interi positivi, che scriva in output tutte le possibili liste di elementi delle liste di *<sistema>* (cioé tutte le possibili colonne del sistema) che abbiano un numero di `x`, di `1` e di `2` rispettivamente minori o uguali a `MaxX`, `Max1` e `Max2`

```
?- sviluppo([[1],[2],[1],[x],[1],[1,2],[1,x],[x],[x],[1],[x],[x],[x]],6,7,1).
[1,2,1,x,1,1,x,x,x,1,x,x,x]
[1,2,1,x,1,1,1,x,x,1,x,x,x]
```

**Esercizio 27.**

Se si vuol calcolare la somma di due numeri naturali maggiori del massimo numero rappresentabile sul sistema si potrebbe pensare di rappresentare ogni numero come una lista delle sue cifre. Ciò premesso, scrivere un predicato `diff(N1,N2,N,Segno)` dove `N1` e `N2` sono due numeri interi positivi scritti sotto forma di lista delle cifre componenti, `N` é la differenza tra il maggiore ed il minore dei due e `Segno` vale '+' se `N1`≥`N2`, vale '–' altrimenti, come nell'esempio. `N1` ed `N2` sono instanziati, `N` e `Segno` devono essere instanziati. Devono essere eliminati gli zero superflui.

```
?- diff([2,1,2,5,8],[2,1,2,4,3],N,Segno).
N=[1,5]
Segno=+
?- diff([6,2,5,8],[3,2,1,4,8],N,Segno).
N=[2,5,8,9,0]
```

`Segno=-`

**Esercizio 28.**

Sia dato un dizionario di parole espresso come database di fatti del tipo `p(parola)`, dove `parola` é un atomo e corrisponde ad una parola del dizionario. Scrivere un predicato PROLOG `cruciverba([Riga1,Riga2,Riga3],M)` che usi le parole nel dizionario per costruire un cruciverba tutto pieno `[Riga1,Riga2,Riga3]` di tre righe ed `M` colonne. `M` é instanziato alla chiamata, `Riga1`, `Riga2ratteri alfabetici; tutte le liste devono corrispondere a parole nel dizionario` e `Riga3` sono liste di `M` caratteri alfabetici e le `M` colonne sono liste di tre caratteri alfabetici; tutte le liste devono corrispondere a parole nel dizionario.

**Esercizio 29.**

Un numero complesso può essere espresso in PROLOG mediante una struttura del tipo `n(SR,R,SI,I)`, dove `SR` ed `R` sono rispettivamente il segno ed il modulo della parte reale, `SI` ed `I` sono rispettivamente il segno ed il modulo della parte immaginaria. Scrivere un predicato PROLOG `calc/0` che legga due numeri complessi, l'operazione da eseguire (+ per la somma e – per la sottrazione) e stampi il numero complesso risultante (come da esempio).

```
?- calc
dai il primo numero complesso n(-,3,-,5).
dai il secondo numero complesso n(-,2,+,1).
dai l'operazione da eseguire + .
il risultato é n(-,5,-,4)
yes
?- calc
dai il primo numero complesso n(-,3,-,5).
dai il secondo numero complesso n(-,2,+,1).
dai l'operazione da eseguire - .
il risultato é n(-,1,-,6)
yes
```

**Esercizio 30.**

Un database PROLOG contiene orari di volo espressi come fatti del tipo:

```
da_a(ancona,roma,[ora(8:10,9:30,[lun,mar,sab]),ora(9:30,10:40,giorn),
                  ora(19:30,20:45,[mer,ven,dom])]).
```

Il primo argomento rappresenta la città di partenza, il secondo argomento la città di arrivo ed il terzo é una lista composta da strutture in cui:

- il primo elemento é l'orario di partenza,
- il secondo é l'orario di arrivo,
- il terzo può essere:
  - una lista, nel qual caso rappresenta i giorni in cui il volo viene effettuato,
  - un atomo "giorno" che rappresenta il fatto che il volo é giornaliero.

Gli orari di partenza e di arrivo sono espressi mediante l'uso dell'operatore infisso " : " non predefinito.

Scrivere un predicato PROLOG `viaggio(Part,Arr,Giorno,Itinerario)` che avendo `Part` e `Arr` instanziate rispettivamente alla città di partenza ed alla città di arrivo di un viaggio determini i giorni e gli itinerari possibili per compiere il viaggio, instanziando `Giorno` al giorno del viaggio ed `Itinerario` alla lista dei tratti (con gli orari) da percorrere, utilizzando il funtore "`perc`" come nell'esempio.

```
?- viaggio(ancona,parigi,Giorno,Itinerario).
Giorno= lun
Itinerario = [perc(ancona,8:10,roma,9:30), perc(roma,10:15,milano,12:00),
perc(milano,20:15,parigi,22:10)]
more (y/n)? y
Giorno= lun
Itinerario = [perc(ancona,8:10,roma,9:30), perc(roma,18:00,milano,19:45),
```

```
perc(milano,20:15,parigi,22:10)]
more (y/n)? y
Giorno= lun
Itinerario = [perc(ancona,8:10,roma,9:30), perc(roma,10:50,parigi,13:20)]
more (y/n)? y
yes
```

**Esercizio 31.**

Scrivere un predicato PROLOG `ruota(N,L,L1)` che é vero se `L1` é la lista `L` ruotata verso sinistra `N` volte. Il predicato (uno solo) con `L` instanziata, gestisca tutte le possibilità di istanziazione di `N` ed `L1` (ovvero: sia `N` sia `L` instanziate, `N` instanziata ed `L1` no, `L1` instanziata ed `L` no).

```
?- ruota(3,[1,2,3,4],L).
L=[4,1,2,3]
?- ruota(N,[1,2,3,4],[2,3,4,1]).
N=1
?- ruota(2,[1,2,3,4],[3,4,1,2]).
yes
```

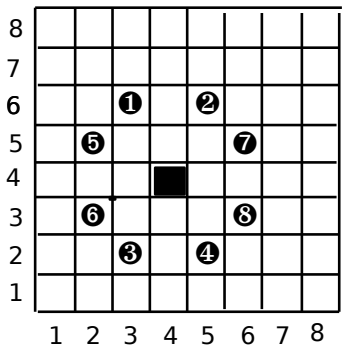
**Esercizio 32.**

Scrivere un predicato PROLOG `sottos(LI,S,LO)` che, con `LI` instanziata ad una lista di interi, tutti maggiori di zero diversi fra loro, ed `S` instanziata ad un numero intero, instanzi `LO`, successivamente, alla lista di interi di `LI` tali che la loro somma sia `S`. In `LO` non deve essere utilizzato più volte lo stesso elemento. Qualora non sia possibile trovare una lista con tale proprietà il predicato fallisca.

```
?- sottos([1,2,3,4,6,8],13,L).
L=[1,2,4,6]
more (y/n)? y
L=[1,4,8]
...
```

**Esercizio 33.**

Determinare la lista ordinata di tutte le possibili posizioni in cui può andare un cavallo su una scacchiera 8x8, con una sola mossa, partendo da una posizione data e compatibilmente con i limiti della scacchiera. La posizione deve essere espressa mediante l'uso dell'operatore infisso `/` (ovvero *RIGA/COLONNA*). La lista deve essere ordinata preferenzialmente rispetto alle righe. Nell'ordinamento si utilizzi il metodo del quicksort. Le possibili mosse del cavallo da una posizione generica sono riportate in figura.



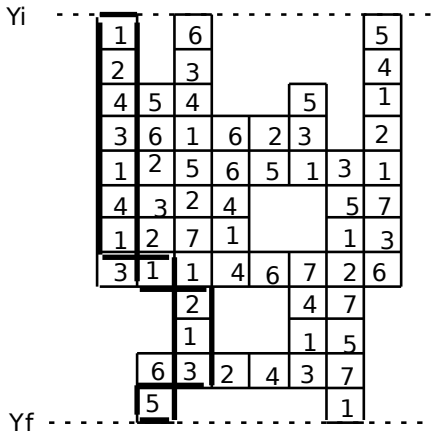

Si supponga predefinito il predicato *findall*.

```
?- posizione(2/3,P).
P=[1/1,1/5,3/1,3/5,4/2,4/4]
?- posizione(8/7,P).
P=[6/6,6/8,7/5]
```

**Esercizio 34.**

Data una matrice sparsa 8-connessa (si può passare da un elemento ad uno qualunque degli otto elementi adiacenti) del tipo rappresentato in figura, darne una conveniente rappresentazione in un database PROLOG. Scrivere un predicato PROLOG `vai(Yi,Yf,Max,Percorso)`

che istanzi `Percorso` successivamente ad ogni possibile percorso di elementi adiacenti, dalla quota `Yi` alla quota `Yf` tale che:
1. la somma dei valori incontrati durante il percorso sia inferiore a `Max`,
2. non si sia passati più di una volta nello stesso elemento.




```
?- vai(12,1,29,P).
P=[r(1,12),r(1,11),r(1,10),r(1,9),r(1,8),r(1,7),r(1,6),r(2,5),r(3,4),r(3,3),r(3,2),r(2,1)]
more? y/n
...
```

**Esercizio 35.**

Scrivere un predicato Prolog `somma(P1,P2,P3)` dove `P1`, `P2` e `P3` sono polinomi (espressi come descritto nel seguito), che è vero se `P3` é la somma di `P1` e `P2`. I polinomi sono espressi mediante una lista di termini del tipo `x(S,C,P)` (corrispondenti a `S C xP`) dove `P` é un intero non negativo corrispondente all'esponente del monomio, `S` e `C` rappresentano il coefficiente, essendo `C` un intero positivo ed `S` il segno (`+` o `-`). I polinomi sono ordinati secondo la potenza crescente. Quando il coefficiente di un termine é nullo il termine stesso non deve comparire nell'espressione del polinomio.

```
?- somma([x(+,3,1),x(-,8,3),x(+,10,4)],[x(+,3,2),x(+,8,3),x(-,7,4),x(+,2,5)],L).
L = [x(+,3,1),x(+,3,2),x(+,3,4),x(+,2,5)]
more (y/n)? y
no
```

**Esercizio 36.**

Si supponga presente nel DB PROLOG un fatto `password(NOME)`, dove `NOME` é instanziata e rappresenta una parola d'ordine espressa come atomo PROLOG. Si scriva un predicato `start` che chieda la password e si comporti come segue:
- qualora venga battuta la password corretta esca con successo
- qualora venga battuta una password contenente uno dei seguenti errori
    1. una lettera di troppo,
    2. una lettera in meno,
    3. una sola lettera errata,
    4. due lettere adiacenti scambiate di posto,

dopo aver segnalato il tipo di errore, richieda di nuovo la password fino ad un massimo di tre volte
- in ogni altro caso esca con fallimento.

Es: supponiamo presente nel DB il fatto `password(amico)`:

```
?- start.
Password ?
amico.
yes
?- start.
Password ?
amoco.
Diversa una lettera.
Password ?
```

```
amco.
La password ha una lettera in più.
Password ?
aimco.
Due lettere adiacenti sono invertite.
no
?- start.
Password ?
amicone.
no
```

Soluzione 36.

```
password(amico).

start:-
      start0(1).
start0(N):-
      N =< 3,
      write('Password ?'),
      read(X),
      name(X,X1),
      password(Y),
      name(Y,Y1),
      start00(N,X1,Y1).

start00(_,X1,Y1):-
      match(X1,Y1),
      !.
start00(N,X1,Y1):-
      match1(X1,Y1),
      !,
      N1 is N + 1,
      start0(N1).

match([],[]).
match([X|Y],[X|Z]) :-
      match(Y,Z).

match1([X,Y|Z],[Y,X|Z]) :-
      write('Due lettere adiacenti sono invertite.').
match1([T1|X],[T2|X]) :-
      T1 \= T2,
      write('Diversa una lettera.').
match1([_|X],X) :-
      write('La password ha una lettera in meno').
match1(X,[_|X]) :-
      write('La password ha una lettera in piu' ).
match1([X1|Y],[X2|Z]) :-
      X1 = X2,
      match1(Y,Z).
```

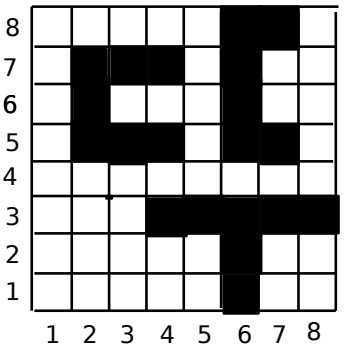
**Esercizio 37.**

Scrivere un predicato PROLOG `calcola/0` che legga da tastiera un'espressione formata da due numeri complessi (con le possibili operazioni +, -, *), e la calcoli come nell'esempio sotto riportato. (Suggerimento: si definisca *i* come operatore ...). Scrivere inoltre un predicato `quick(L,LORD)` che, utilizzando il metodo del quicksort ordini la lista `L` *di* numeri complessi secondo valori crescenti del modulo.

```
?- calcola.
dai l'espressione (4+ i 7) - (2+ i 3).
il risultato é  2+ i 4
?- quick([1+i 2),(3+i 4), (2+i 1), (5+i 3)],L).
L=[2+i 1, 1+i 2, 3+i 4), 5+i 3]
More (y/n)? y
no
```

**Esercizio 38.**

Data una matrice di pixel del tipo di quella presentata in figura,




darne una possibile rappresentazione. Scrivere inoltre i seguenti predicati:

- `u(X)` che istanzi `X` alla lista `[p(X1,Y1),p(X1,Y2),p(X2,Y2),p(X2,Y1)]` dove le coppie `p(Xi,Yi)` rappresentano gli spigoli di una `U` in senso antiorario; `p(X1,Y1)` e `p(X2,Y1)` sono gli estremi della `U`;
- `c(X)` che istanzi `X` alla lista `[p(X1,Y1),p(X2,Y1),p(X2,Y2),p(X1,Y2)]` dove le coppie `p(Xi,Yi)` rappresentano gli spigoli di una `C` in senso antiorario; `p(X1,Y1)` e `p(X1,Y2)` sono gli estremi della `C`;
- `t(X)` che istanzi `X` alla lista `[p(X1,Y1),p(X2,Y2),p(X3,Y1)]` dove le coppie `p(Xi,Yi)` rappresentano i punti estremi di una `T` in senso antiorario. Le lettere s'intendono composte di soli segmenti orizzontali o verticali.

```
?- c(X).
X = [p(4,7),p(2,7),p(2,5),p(4,5)] -> ;
X = [p(7,8),p(6,8),p(6,5),p(7,5)] -> ;
no
?- t(X).
X = [p(4,3),p(6,1),p(8,3)] -> ;
no
```

**Esercizio 39.**

Scrivere un predicato PROLOG `uguali(Li,Lo)` che data una lista `Li` di parole (espresse come atomi) restituisca la lista `Lo` delle parole che iniziano e terminano con una stessa lettera. La lista deve essere in ordine alfabetico rispetto alla sola prima lettera. A parità di prima lettera siano date in ordine di lunghezza decrescente. Per l'ordinamento si utilizzi il metodo del mergesort.

```
?- uguali([aa,io,amica,paio,pop,non],Lo).
Lo = [amica,aa,non,pop] -> ;
no
```

**Esercizio 40.**

In riferimento alla figura, scrivere un predicato PROLOG `coppia(L,A,B)`, dove `L` é istanziata ad una lista di archi, che, dopo aver verificato che la lista indichi un cammino corretto per andare esattamente da un nodo iniziale (identificato con un quadrato) ad un nodo finale (identificato con un triangolo), istanzi `A` e `B` rispettivamente al nodo iniziale ed al nodo finale. Qualora il cammino non sia corretto venga stampato "percorso non corretto" e vengano istanziate `A` e `B` a `nil`.

```
?- coppia([a,d,g,i],A,B).
A=1
B=11
yes
```

**Esercizio 41.**

Scrivere un programma PROLOG che esegua la somma di una lista di un numero qualsiasi di numeri interi espressi a loro volta come lista delle cifre componenti.

```
?- somma([[2,3,6],[4,1,2,8],[5,0]],N).
N=[4,4,1,4]
```

**Esercizio 42.**

Scrivere un predicato PROLOG `anagramma(Parola, Anagr)` in cui `Parola` é istanziata ad una parola (di tutte lettere minuscole), che istanzi `Anagr`, per risoddisfacimento, a tutti gli anagrammi di `Parola`. Si deve evitare di restituire soluzioni in cui siano presenti più di tre consonanti consecutive o più di due vocali consecutive.

```
?- anagramma(vincerei, Anagr).
Anagr = vincerei -> ;
Anagr = vincerie -> ;
Anagr = vinceeri -> ;
Anagr = vinceire -> ;   (N.B.: scartato vinceeir)
Anagr = vincerei -> ;   (N.B.: scartato vinceier)
Anagr = vinceeri -> ;
...
```

**Esercizio 43.**

Scrivere un predicato PROLOG `run(L)`, dove `L` é una lista di termini qualsiasi (composti e non), che, scartando i termini atomici ed i termini contenenti argomenti non atomici, ordini il resto in modo alfabeticamente crescente (secondo il codice ASCII) rispetto al funtore (indipendentemente dalla sua arità) e ne dia una stampa ordinata, secondo l'esempio che segue.

```
?- run([prova(2,1),term(4),a(g(7)),tizio(1),pinco(z,t,v),quit,per(i,i)]).

 FUNTORE                    ARGOMENTI
 per                        i i
 pinco                      z t v
 prova                      2 1
 term                       4
 tizio                      1
```

**Esercizio 44.**

Sia dato un database PROLOG di fatti del tipo `p(`PAROLA`)` dove PAROLA é un atomo corrispondente ad una parola della lingua italiana. Scrivere un predicato PROLOG `rima`, senza argomenti, che legga dalla tastiera una parola e stampi tutte le parole presenti del database facenti rima con la stessa. La stampa deve avvenire separatamente, partendo da quelle che fanno rima solo per l'ultima lettera, proseguendo con quelle che fanno rima solo per le ultime due lettere etc., fino a quelle che fanno rima per tutta la parola data. La stampa delle parole facenti rima per uno stesso numero di lettere deve essere effettuata in ordine alfabeticamente crescente secondo lo schema riportato nell'esempio. Per l'ordinamento si utilizzi il metodo del quicksort.
Esempio di database:

```
p(pino). p(pippo). p(vino). p(tonno). p(ceppo). p(tino).
p(ceppi). p(parigino). p(ticino). p(micino). p(camicino). p(gino). p(cugino).

?- rima.
dai la parola
gino.

rime di n. 1 lettere
ceppo
pippo

rime di n. 2 lettere
tonno

rime di n. 3 lettere
```

```
camicino
micino
pino
ticino
tino
vino

rime di n. 4 lettere
cugino
gino
parigino
```

**Esercizio 45.**

Sia dato un Database PROLOG contenente le informazioni di un insieme di famiglie italiane organizzato in fatti con la struttura mostrata nell'esempio. Sono riportati: nome, cognome, data di nascita, lavoro e reddito annuo (espresso in milioni di lire o l'atomo `disoccupato` nel caso non lavori) del capofamiglia, della moglie e, nella lista, quelli dei figli. Scrivere un predicato PROLOG `run` (senza argomenti) che stampi, secondo il formato dell'esempio, per tutte le famiglie il cognome e il nome del capofamiglia, il reddito medio pro-capite della famiglia e il numero dei figli, in ordine descrescente rispetto al reddito medio pro-capite. Alla fine stampi il reddito medio pro-capite di tutto l'insieme considerato e il numero medio di figli per famiglia. Non deve essere utilizzato il predicato di sistema `setof` e per l'ordinamento si usi il metodo del Quicksort.

```
famiglia(
persona(rossi,andrea,data(8,4,1947),lavoro(olivetti,80.5)),
persona(bianchi,anna,data(18,5,1947),lavoro(fiat,51)),
[persona(rossi,franco,data(3,12,1969),lavoro(usl,52.7)),
persona(rossi,ileana,data(1,6,1982),disoccupato)]).

famiglia(
persona(baldi,aldo,data(18,4,1954),lavoro(bnl,81)),
persona(andreani,adriana,data(1,2,1957),disoccupato),
[]).
.................................

famiglia(
persona(testi,alfio,data(9,1,1948),disoccupato),
persona(bianchi,anna,data(1,5,1952),disoccupato),
[persona(testi,franco,data(13,2,1978),disoccupato)]).

famiglia(
persona(conti,andrea,data(6,4,1949),disoccupato),
persona(bianchi,nadia,data(1,5,1952),lavoro(fiat,54.2)),
[persona(conti,franco,data(13,2,1976),disoccupato),
persona(conti,ileana,data(14,6,1980),disoccupato),
persona(conti,dario,data(4,11,1985),disoccupato)]).

famiglia(
persona(gobbi,andrea,data(16,5,1942),lavoro(municipio,60.5)),
persona(ferrari,adelia,data(21,5,1944),lavoro(municipio,53.2)),
[persona(gobbi,francesco,data(1,2,1968),lavoro(bnl,55.2)),
persona(gobbi,ilde,data(1,5,1970),lavoro(fiat,50.1)),
persona(gobbi,dante,data(14,1,1980),disoccupato)]).

?- run.
COGNOME NOME      REDD.M.     N. FIGLI
rossi andrea      46.05 2
gobbi andrea      43.8        3
baldi aldo        40.05 0
.....
conti andrea      10.84 3
testi alfio       O.O         1
Reddito medio pro capite 28.238
Numero medio di figli per famiglia 1.8
```

**Esercizio 46.**

Un albero 2-3-4 o é vuoto o consiste di una radice e di 2 o 3 o 4 sottoalberi. Scrivere un predicato PROLOG `stampa(ALBERO)` dove `ALBERO` viene istanziato ad un albero 2-3-4 con functore `alb`, che stampi graficamente l'albero con la radice a sinistra e le foglie a destra, i sottoalberi sinistri in alto ed i sottoalberi destri in basso secondo i seguenti quattro casi:

0. caso di un solo sottoalbero

```
                sottoalbero1
    radice
```

1. caso di due sottoalberi

```
                sottoalbero1
    radice
                sottoalbero2
```

2. caso di tre sottoalberi

```
                sottoalbero1
                sottoalbero2
    radice
                sottoalbero3
```

3. caso di tre sottoalberi

```
                sottoalbero1
                sottoalbero2
    radice
                sottoalbero3
                sottoalbero4
```

con interlinea singolo. La spaziatura da sx a dx viene chiesta in input.

```
?-
stampa(alb(1,alb(11,alb(111,n,n),alb(112,alb(1121,n,n),n)),alb(12,alb(121,n,n),alb(122,n,n),
alb(123,n,n),alb(124,alb(1241,n,n),alb(1242,n,n)))))

Di quanti spazi vuoi indentare?
15.
                        111
            11
                                    1121
                        112
1
                        121
                        122
            12
                        123
                                    1241
                        124
                                    1242
yes
```

**Esercizio 47.**

Scrivere un predicato PROLOG `run(L)` che chieda in input un carattere alfanumerico alla volta e termini con successo quando sono premuti nella sequenza esatta i caratteri contenuti in una lista argomento del predicato `codice(Lista)` asserito come fatto. In caso di successo restituisca la lista `L` instanziata alla lista dei caratteri immessi ordinata in modo crescente. Se sono premuti più di dieci caratteri il programma termini comunque con un fallimento. Nell'ordinamento si faccia uso del me-

todo del Merge-sort.
Es: avendo impostato nel Database il seguente fatto

```
codice([0,a,b]).

?-run(L).
0.
a.
c.
0.
a.
b.
L=[a,a,b,c,0,0] ->;
no

?-run(L).
0.
a.
c.
a.
2.
7.
b.
a.
6.
c.
no
```

**Esercizio 48.**

Scrivere un predicato PROLOG `run(L1,L2)`, che legga da input caratteri fino a quando non venga immesso un punto (`.`), istanzi `L1` alla lista dei caratteri alfabetici incontrati ed `L2` alla lista dei caratteri numerici. Le liste `L1` ed `L2` vengano restituite prive di caratteri ripetuti (due caratteri uguali contigui).

**Esercizio 49.**

Scrivere un predicato PROLOG `run(N,X)`, dove `N` è un numero intero positivo instanziato alla chiamata e `X` una lista da istanziarsi in uscita, che realizzi quanto descritto in seguito.
Chieda all'utente, fino a quando non viene battuto `basta`, alcune cifre pari (zero compreso) che vengono conservate nel Data Base PROLOG.
Chieda all'utente, fino a quando non viene battuto `basta`, alcune cifre dispari che vengono conservate nel Data Base PROLOG.
Istanzi `X` alla lista (ordinata in modo crescente) dei numeri di quattro cifre maggiori di `N` che possono essere formati usando le cifre precedentemente immesse, alternando cifre pari e cifre dispari. Deve essere controllato che:
- le cifre immesse come pari lo siano effettivamente, ed analogamente per quelle dispari;
- una cifra non venga immessa più di una volta;
- i numeri formati con le suddette cifre non contengano zeri iniziali.

Si preveda la riutilizzazione del predicato `run(N,X)` nella stessa sessione di lavoro dell' interprete. Qualora si intenda usare un algoritmo di ordinamento si usi quello dell' *insertion-sort*.

**Esercizio 50.**

Sia dato un database di fatti del tipo `connesso(L1,L2,V,D)`, dove `L1` e `L2` sono due luoghi geografici, `V` é la via che li collega e `D` é la distanza che li separa. Scrivere un programma PROLOG `vai(Partenza,Arrivo,Percorso,Lunghezza)` che, dati i luoghi di `Partenza` e di `Arrivo`, fornisca, per risoddisfacimento, tutti i possibili percorsi `Percorso` fra i due (senza passare più di una volta nello stesso luogo), con la relativa lunghezza `Lunghezza`, a partire da quello a lunghezza minima, via via fino a quello a lunghezza massima. Un percorso non é rappresentato con una lista di luoghi ma con una lista di strutture `p(V,D)`, dove `V` é la via che collega due luoghi e `D` é la distanza che li separa.

Nel caso in cui nel percorso compaiano di seguito *n* strutture con la stessa via
`...,p(V,D1),p(V,D2),...,p(V,Dn),...`
compattarle tutte in un'unica struttura `p(V,D)` in cui $D=\sum_{i=1}^{n} D_i$

Nel caso si intenda usare un metodo di ordinamento, si utilizzi quello del quicksort.
Esempio. Si consideri il seguente database:

```
connesso(passetto,dorico,viale_della_vittoria,3).
connesso(dorico,passetto,viale_della_vittoria,3).
connesso(piazza_diaz,dorico,viale_della_vittoria,3).
connesso(piazza_diaz,piazza_cavour,viale_della_vittoria,7).
connesso(piazza_cavour,piazza_repubblica,corso_garibaldi,5).
connesso(piazza_repubblica,s2,via_29_settembre,7).
connesso(s2,piazza_italia,viale_marconi,8).
connesso(piazza_italia,piazza_ugo_bassi,corso_carlo_alberto,7).
connesso(piazza_liberta,piazza_europa,via_martiri_resistenza,6).
connesso(piazza_cavour,piazza_liberta,galleria,9).
connesso(piazza_liberta,piazza_europa,via_martiri_resistenza,6).
connesso(piazza_europa,piazza_ugo_bassi,via_ricostruzione,4).
connesso(piazza_ugo_bassi,pinocchio,via_maggini,14).

?_ vai(passetto,pinocchio,P,D).

P = [p(viale_della_vittoria, 13), p(galleria, 9), p(via_martiri_resistenza, 6),
p(via_ricostruzione, 4), p(via_maggini, 14)]
D = 46 -> ;

P = [p(viale_della_vittoria, 13), p(corso_garibaldi, 5), p(via_29_settembre, 7),
p(viale_marconi, 8), p(corso_carlo_alberto, 7), p(via_maggini, 14)]
D = 54
yes
```

**Esercizio 51.**

Si consideri una qualsiasi formula proposizionale contenente i soli connettivi logici di negazione, congiunzione e disgiunzione. Avendo predefinito i seguenti operatori:

```
:-  op(30,fx,non).
:-  op(100,xfy,or).
:-  op(100,xfy,and).
```

rappresentiamo detta formula proposizionale mediante una struttura Prolog in cui le lettere proposizionali sono rappresentate da atomi Prolog (es.: `(a or non (b and c))`)
Scrivere un programma Prolog `tabella(FP)` che prenda in `FP` una struttura Prolog rappresentante una formula proposizionale e, elencate le *n* lettere proposizionali distinte che vi compaiono, scriva in output la sua tabella di verità; detta tabella é costituita da $2^n$ righe di due elementi: il primo elemento é la lista delle possibili assegnazioni di valore di verità `v` o `f` alle *n* lettere proposizionali contenute nella formula, il secondo elemento é il valore di verità risultante di tutta la formula.
Si supponga predefinito un predicato `substitute(Lp,V,Fp,Fpv)` in cui `Lp` é un atomo (lettera proposizionale), `V` é un atomo (`v` o `f`), `Fp` é una struttura (formula proposizionale) e `Fpv` e' la struttura `Fv` in cui ogni occorrenza
dell'atomo `Lp` e' stata sostituita dall'atomo `V`.

```
:-  tabella(a and (b or non a))

[a, b]
[v, v]  v
[v, f]  f
[f, v]  f
[f, f]  f
yes


:-  tabella(a and (c or b or non a))
```

```
[a, b, c]
[v, v, v]  v
[v, v, f]  v
[v, f, v]  v
[v, f, f]  f
[f, v, v]  f
[f, v, f]  f
[f, f, v]  f
[f, f, f]  f
yes
```

**Soluzione 51.**

```
valore(f, f) :- !.
valore(v, v) :- !.

valore(non v, f) :- !.              % tabella di verità del connettivo non
valore(non f, v) :- !.              %

valore(v and v, v) :- !.            %tabella di verità del connettivo and
valore(v and f, f) :- !.            %
valore(f and v, f) :- !.            %
valore(f and f, f) :- !.            %

valore(v or v, v) :- !.             % tabella di verità del connettivo or
valore(v or f, v) :- !.             %
valore(f or v, v) :- !.             %
valore(f or f, f) :- !.             %

valore(non A,X) :-            % determinazione del valore di verità
      valore(A,A1),           % della formula non A, dove A é una
      valore(non A1,X).       % formula proposizionale

valore(A and B,X) :-          % determinazione del valore di verità
      valore(A,A1),           % della formula A and B, dove A e B
      valore(B,B1),           % sono formule proposizionali
      valore(A1 and B1,X).    %

valore(A or B,X) :-           % determinazione del valore di verità
      valore(A,A1),           % della formula A or B, dove A e B
      valore(B,B1),           % sono formule proposizionali
      valore(A1 or B1,X).     %

foglia(F,F) :-
       atomic(F).
foglia(F,L) :-
       not  atomic(F),
       F =.. [Funtore|Argomenti],
       member(X,Argomenti),
       foglia(X,L).

proposizioni(F,L) :-
       setof(X,foglia(F,X),L).

member(X,[X|_]).
member(X,[Y|Z]) :- member(X,Z).

cambia([T|C1],[X|C2]) :-
      member(X,[v,f]),
      cambia(C1,C2).
cambia([],[]).

tabella(F):-                  % predicato principale
      proposizioni(F,L),      % cerca le lettere proposizionali
      nl, write(L),           % le stampa a mo' d'intestazione
      cambia(L,L1),           % genera una combinazione di valori di verità
      sostituisci(F,L,L1,V),  % calcola il valore di verità della formula
      nl,write(L1),write('  '),write(V),
      fail.
tabella(_).
```

```
sostituisci(F,[T1|C1],[T2|C2],X) :-
      substitute(T1,T2,F,NF),
      sostituisci(NF,C1,C2,X).
sostituisci(NF,[],[],X) :-
      valore(NF,X).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 :-  op(30,fx,non).
 :-  op(100,xfy,or).
 :-  op(100,xfy,and).

substitute(Old,New,Old,New) :- !.
substitute(Old,New,Term,Term):-
       atomic(Term),
       Term \= Old.
substitute(Old,New,Term,Term1):-
       not(atomic(Term)),
       functor(Term,F,N),
       functor(Term1,F,N),
       substitute(N,Old,New,Term,Term1).

substitute(N,Old,New,Term,Term1):-
       N > 0,
       arg(N,Term,Arg),
       substitute(Old,New,Arg,Arg1),
       arg(N,Term1,Arg1),
       N1 is N - 1,
       substitute(N1,Old,New,Term,Term1).
substitute(0,Old,New,Term,Term1).
```

**Esercizio 52.**

Scrivere un programma PROLOG `esamina(L1,L2)`, dove `L1` é istanziata ad una lista di atomi ed `L2` deve essere istanziata in uscita alla lista di strutture `int(I,F)`, in cui `I` é un intero appartenente ad `L1` ed `F` é il numero delle volte che lo stesso appare in `L1`. La lista `L2` deve essere ordinata in modo decrescente rispetto alla frequenza `F` e, a paritaà di frequenza, in modo crescente rispetto ad `I`.
L'esecuzione del predicato produca come side-effect l'asserimento di `L2` nel database Prolog come fatto `lista_f(L2)`.
Scrivere inoltre un predicato `totale(A)` che fonde le frequenze contenute nei fatti `lista_f(L)` fino a quel momento asseriti, nell'unica lista `A` ordinata secondo gli stessi criteri di `L2` del precedente predicato esamina. Qualora non sia presente alcun fatto `lista_f` venga restituito `nil`. Dopo l'esecuzione del predicato `totale(A)` il database non deve contenere più fatti `lista_f`. Per l'ordinamento si faccia uso del metodo del quicksort.

**Esercizio 53.**

Scrivere un predicato PROLOG `run(N)` che chieda da tastiera, finché non viene battuto `basta`, delle cifre e che restituisca, in risoddisfacimento, tutti i numeri che possono essere formati con le cifre immesse, i quali siano divisibili per 3 e non lo siano per 4, in ordine decrescente.
Per il controllo di divisibilità si faccia riferimento ai "criteri di divisibilità" dell'aritmetica elementare.
Non devono essere utilizzati predicati che modificano il Data Base.
Le cifre immesse non possono essere ripetute (saranno quindi immesse al massimo dieci cifre).
Qualora non siano state immesse cifre, `N` venga istanziato a `nil`.

```
?-run(N).
dai una cifra __ 4.
dai una cifra __ 8.
dai una cifra __ 0.
dai una cifra __ 4.
cifra ripetuta; dai una cifra __ 3.
```

```
dai una cifra __ basta.

N = 8883 -> ;
N = 8838 -> ;
N = 8433 -> ;
N = 8430 -> ;
N = 8403 -> ;
. . . . . .

N = 303 -> ;
N = 33 -> ;
N = 30 -> ;
N = 3 -> ;
no
```

**Esercizio 54.**

Si considerino i tre simboli del totocalcio `1`, `x` e `2`. Sia `<partita>` un tipo di lista contenente un simbolo solo (giocata fissa) o due (doppia) o tutt'e tre (tripla). Sia `<sistema>` un tipo di lista di (eventualmente 13) liste del tipo `<partita>`. Si definisca un predicato
`sviluppo(S,MaxX,Max1,Max2,MaxseqX,Maxseq1,Maxseq2,Costo_Colonna)`
dove `S` è una lista di tipo `<sistema>` e gli altri argomenti sono interi positivi, che scriva in output tutte le possibili liste costruite prendendo un elemento per ciascuna lista di tipo `<partita>` di `S` (cioé tutte le possibili colonne del sistema) che abbiano un numero di `1`, `x` e `2` rispettivamente minori o uguali a `Max1`, `MaxX` e `Max2`, nonché presentino sequenze di `1`, `x` e `2` rispettivamente non più lunghe di `Maxseq1`,`MaxseqX` e `Maxseq2`. Alla fine si stampi il costo del sistema (numero delle colonne * `Costo_Colonna`).

```
?-  sviluppo([[1, x], [1], [2, x], [1, x], [x], [x], [x], [x], [x], [1, x], [1], [2, x], [1,
x]], 8, 4, 2, 13, 13, 13, 400)

[1, 1, 2, 1, x, x, x, x, x, x, 1, 2, x]
[1, 1, 2, 1, x, x, x, x, x, x, 1, x, x]
[1, 1, 2, x, x, x, x, x, x, 1, 1, 2, x]
[1, 1, 2, x, x, x, x, x, x, 1, 1, x, x]
[1, 1, 2, x, x, x, x, x, x, x, 1, 2, 1]
[1, 1, 2, x, x, x, x, x, x, x, 1, 2, x]
[1, 1, 2, x, x, x, x, x, x, x, 1, x, 1]
[1, 1, x, 1, x, x, x, x, x, x, 1, 2, x]
[1, 1, x, x, x, x, x, x, x, 1, 1, 2, x]
[1, 1, x, x, x, x, x, x, x, x, 1, 2, 1]
[x, 1, 2, 1, x, x, x, x, x, 1, 1, 2, x]
[x, 1, 2, 1, x, x, x, x, x, 1, 1, x, x]
[x, 1, 2, 1, x, x, x, x, x, x, 1, 2, 1]
[x, 1, 2, 1, x, x, x, x, x, x, 1, 2, x]
[x, 1, 2, 1, x, x, x, x, x, x, 1, x, 1]
[x, 1, 2, x, x, x, x, x, x, 1, 1, 2, 1]
[x, 1, 2, x, x, x, x, x, x, 1, 1, 2, x]
[x, 1, 2, x, x, x, x, x, x, 1, 1, x, 1]
[x, 1, 2, x, x, x, x, x, x, x, 1, 2, 1]
[x, 1, x, 1, x, x, x, x, x, 1, 1, 2, x]
[x, 1, x, 1, x, x, x, x, x, x, 1, 2, 1]
[x, 1, x, x, x, x, x, x, x, 1, 1, 2, 1]
Costo totale = 8800
yes

?-  sviluppo([[1, x], [1], [2, x], [1, x], [x], [x], [x], [x], [x], [1, x], [1], [2, x], [1,
x]], 8, 4, 2, 6, 1, 13, 400)

[x, 1, 2, 1, x, x, x, x, x, x, 1, 2, 1]
[x, 1, 2, 1, x, x, x, x, x, x, 1, 2, x]
[x, 1, 2, 1, x, x, x, x, x, x, 1, x, 1]
[x, 1, x, 1, x, x, x, x, x, x, 1, 2, 1]
Costo totale = 1600
yes
```

**Soluzione 54.**

```
sviluppo(X,Mx,M1,M2,MaxSeqx,MaxSeq1,MaxSeq2,C_Unit) :-
      sistema(X,Mx,M1,M2,Col),
      maxseq(Col,x,MaxSeqx),
      maxseq(Col,1,MaxSeq1),
      maxseq(Col,2,MaxSeq2),
      write(Col),nl,aggiorna,fail.
sviluppo(_,_,_,_,_,_,_,C_Unit) :- numero(N), Tot is N * C_Unit,
      write('Costo totale '),nl.

sistema(X,MaxX,Max1,Max2,Col):-
      colonna(X,Col),
      conta(Col,x,Nx),
      conta(Col,1,N1),
      conta(Col,2,N2),
      Nx =< MaxX, N1 =< Max1, N2 =< Max2.

aggiorna :- retract(numero(N)),!,N1 is N +1,assert(numero(N1)).
aggiorna :- assert(numero(1)).

colonna([],[]).
colonna([X|Z],[Y|Col]) :-
      member(X,Y),
      colonna(Z,Col).

conta([],_,0).
conta([X|Y],X,NX) :-  conta(Y,X,N), NX is N+1.
conta([Z|Y],X,NX) :-  Z \== X, conta(Y,X,NX).

maxseq(Col,Segno,Numero) :-
      maxseq0(Col,Segno,0,Numero).
maxseq0([],_,_,_).
maxseq0(C,T,Cont,Numero) :-
      Cont > Numero, !, fail.
maxseq0([T|C],T,Cont,Numero) :-
      NCont is Cont + 1,
      maxseq0(C,T,NCont,Numero).
maxseq0([T1|C],T,Cont,Numero) :-
      T1 \== T,
      maxseq0(C,T,0,Numero).

member([X|_],X).
member([_|X],Y) :- member(X,Y).
```

**Esercizio 55.**

Si suppongano presenti nel DB PROLOG fatti `diz(PAROLA, WORD)`, in cui `PAROLA` e' un atomo rappresentante una parola italiana e `WORD` la corrispondente parola in lingua inglese.
Scrivere un predicato PROLOG *traduci* che legga da tastiera, fino a quando non viene immessa la lista vuota, una frase (espressa come lista di atomi) in italiano e stampi la corrispondente in inglese utilizzando i fatti `diz` del DB secondo lo schema dell'esempio sotto riportato.
Qualora una parola italiana non sia presente ne DB venga seguito il seguente schema
i) qualora esistano parole italiane della stessa lunghezza che differiscono per una sola lettera, si scelga come parola inglese quella corrispondente alla parola italiana avente la lettera diversa più vicina (secondo il codice ASCII);
ii) qualora non sia verificato il punto precedente, se esistono parole italiane della stessa lunghezza che differiscono per solo due lettere contigue, si scelga come parola inglese quella corrispondente alla parola italiana avente la minima distanza totale dalle lettera diverse (secondo il codice ASCII);
iii) qualora non siano verificati nessuno dei punti precedenti venga tradotta con una parola della stessa lunghezza e composta da tutte lettere uguali a quella iniziale.
Se la distanza e' uguale se ne scelga una qualsiasi. Ad esempio, si supponga che il DB PROLOG sia il seguente

```
diz(una, a).
```

```
diz(bella,beautiful).   diz(beffa,derision).   diz(bolla,bubble).
diz(pianta,tree).       diz(fianco,side).
```

La seguente è una possibile interazione

```
?-traduci.
dai una frase
[una,bella,pianta].
[a,beautiful,tree]

dai un'altra frase
[una,billa,gianca].
[a,beautiful,gggggg]

dai un'altra frase
[una,bezza,ianta].
[a,beautiful,iiiii]

dai un'altra frase
[].

yes
```

**Esercizio 56.**

Due letterali ground si dicono *complementari* se uno è un atomo e l'altro è la negazione di quello stesso atomo. Siano $C_1$ e $C_2$ due clausole (*clausole genitrici*). Se esiste un letterale $l_1$ in $C_1$ che risulti complementare ad un letterale $l_2$ in $C_2$, allora il *risolvente* di $C_1$ e $C_2$ è la clausola $R_{1,2}$ (=$R_{2,1}$) ottenuta dalla disgiunzione di tutti i letterali di $C_1$ e $C_2$ meno $l_1$ ed $l_2$. Si ricorda che l'algoritmo base per determinare l'insoddisfacibilità di un insieme S di clausole mediante la risoluzione può essere scritto come segue.

```
INSIEME_DI_CLAUSOLE := S;
while  "←" non è membro di INSIEME_DI_CLAUSOLE do
     begin
     seleziona in INSIEME_DI_CLAUSOLE due clausole C_i e C_j
          distinte e risolvibili;
     calcola un risolvente R_ij di C_i e C_j che non appartenga ad
INSIEME_DI_CLAUSOLE;
     INSIEME_DI_CLAUSOLE := INSIEME_DI_CLAUSOLE ∪ {R_ij}
     end.
```

L'idea della risoluzione lineare è quella di utilizzare, come una delle due clausole da risolvere, il risolvente calcolato al passo precedente. L'altra clausola può essere o una di quelle di partenza oppure un risolvente calcolato in qualche passo precedente.

Si supponga predefinito l'operatore: `?-op(30,fx,non).` Si rappresenti i letterali positivi (ground) come atomi PROLOG ed i letterali negativi come strutture PROLOG con funtore `non`. Si rappresentino le clausole (ground) come liste di letterali. Si scriva un programma PROLOG con predicato principale `refuta_lin(S)`, dove `S` é una lista di clausole, che applichi il metodo della risoluzione lineare terminando con successo se viene prodotta la clausola vuota (`[]`), terminando con fallimento se al passo corrente non esistono due clausole distinte e risolvibili che producano un risolvente mai generato prima. Prima di introdurre un nuovo risolvente nell'insieme di clausole se ne eliminino le eventuali ripetizioni di letterali. Ad ogni passo di risoluzione si scrivano in output le due clausole genitrici ed, a capo, il risolvente calcolato.

```
?-  refuta_lin([[non a, non b], [non a, b], [a, non b], [a, b]])

[non a, non b]        [non a, b]
[non a]
[non a]        [a, non b]
[non b]
[non b]        [a, b]
```

```
[a]
[a]         [non a, b]
[b]
[b]         [non b]
[]
yes
```

**Soluzione 56.**

```
:-  op(30,fx,non).

refuta_lin(Clausole) :-
      member(C,Clausole),
      refuta(Clausole,C), !.

refuta(Clausole, R) :-
      R \= [],
      member(C,Clausole),
      risolvi(R,C,NR),
      elim_doppi(NR,NR1),
      not member(NR1,Clausole),
      write(R),tab(10), writenl(C), nl,
      append([NR1],Clausole,NuovoClausole),
      refuta(NuovoClausole, NR1).
refuta(Clausole, R) :-
      R = [],
      write(R).

risolvi(C1,C2,R) :-           % calcola il risolvente delle due clausole
      member(L1,C1),          % genitrici ground C1 e C2.
      member(L2,C2),          %Se il risolvente non esiste il goal fallisce.
      (L1 = non L2 ; L2 = non L1),
      cancella(L1,C1,NC1),
      cancella(L2,C2,NC2),
      append(NC1,NC2,R).

cancella(L,[L|C],C1) :-
      cancella(L,C,C1).
cancella(L,[X|C],[X|C1]) :-
      L \= X,
      cancella(L,C,C1).
cancella(L,[],[]).

elim_doppi([T|C],[T|C1]) :-
      not member(T,C),
      elim_doppi(C,C1).
elim_doppi([T|C],C1) :-
      member(T,C),
      elim_doppi(C,C1).
elim_doppi([],[]).

member(X,[X|_]).
member(X,[Y|Z]) :- member(X,Z).
```

**Esercizio 57.**

Si supponga di avere nel Data Base PROLOG un insieme di angoli espressi come fatti a due argomenti `ang(ANGOLO,MISURA)`, dove `ANGOLO` è il nome dell'angolo e `MISURA` è il corrispondente valore nel sistema sessagesimale, rappresentato mediante una lista di tre elementi (gradi, primi, secondi).
Scrivere un predicato PROLOG `minima(CA)` che restituisca la coppia di angoli `CA` (espressa mediante l'uso della " / " come operatore infisso tra i due angoli - vedi esempio) che hanno la minima differenza di valore in modulo. Tenendo conto che si considera il valore assoluto, la differenza tra $\alpha$ e $\beta$ è equivalente a quella tra $\beta$ e $\alpha$ e quindi, durante la ricerca, si deve evitare di considerare coppie equivalenti. La parte del Data Base relativo agli angoli non deve essere modificato. Si può supporre che il numero di angoli del Data Base non sia grande. Ad esempio, si suppongano presenti nel Data Base i seguenti fatti

```
ang(alfa,[10,23,18]).
ang(beta,[22,0,38]).
ang(gamma,[12,28,30]).
ang(delta,[21,30,28]).
ang(epsilon,[21,59,58]).
```

Si ha la seguente chiamata e risposta

```
?- minima(CA).
CA = beta / epsilon ->;
no
```

**Soluzione 57.**

```
minimo(CA):-
      abolish(preso/1),
      setof(X,coppia(X),[c(D,CA)|RESTO]).
/*
stampa([c(D,CA)|RESTO]) .
stampa([]).
stampa([T|C]):- write(T),nl,stampa(C).
*/

coppia(c(D,ANG1/ANG2)):-
      ang(ANG1,VAL1),
      assert(preso(ANG1)),
      ang(ANG2,VAL2),
      not preso(ANG2),
/*
      ANG1 \= ANG2,
*/
diff_abs(VAL1,VAL2,D).
diff_abs(N1,N2,N):-
      magg(N1,N2),
      !,
      diff(N1,N2,N).
diff_abs(N1,N2,N):-
      diff(N2,N1,N).

diff(N1,N2,N):-
      reverse(N1,N1R),
      reverse(N2,N2R),
      diff0(N1R,N2R,NR),
      reverse(NR,N).

diff0([],[],[]).
diff0([T1|C1],[T2|C2],[T|C]):-
      T1 >= T2,
      !,
      T is T1 - T2,
      diff0(C1,C2,C).
diff0([T1A,T1B|C1],[T2|C2],[T|C]):-
      T1A < T2,
      !,
      T1BN is T1B - 1,
      T1AN is T1A + 60,
      T is T1AN - T2,
      diff0([T1BN|C1],C2,C).

magg([T1|C1],[T2|C2]):-
      T1 > T2,
      !.
magg([T1|C1],[T2|C2]):-
      T1 = T2,
      magg(C1,C2).

reverse(L1,L2):-
      rev(L1,[],L2).
rev([],L,L).
```

```
rev([T|C],ACC,L):-
      rev(C,[T|ACC],L).

ang(alfa,[10,23,18]).
ang(beta,[22,0,38]).
ang(gamma,[12,28,30]).
ang(delta,[21,30,28]).
ang(epsilon,[21,59,58]).
```

**Esercizio 58.**

Si rappresenti una matrice quadrata $\begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \dots & \dots & \dots \\ a_{n,1} & \dots & a_{n,n} \end{bmatrix}$ come lista di $n$ liste `[[a1,1,...,a1,n],...,[an,1,...,an,n]]`, dove ciascuna lista di $n$ elementi rappresenta una riga. Scrivere un predicato PROLOG `run(Mat,MatTrasp)` che legga da input una matrice, istanzi `Mat` alla matrice letta e istanzi `MatTrasp` alla matrice trasposta (righe per colonne) di quella letta. La lettura in input deve avvenire con la seguente modalità. Il programma chiede inizialmente la dimensione della matrice. Nota la dimensione della matrice il programma provvede ad assegnare il valore "1" ad ogni suo elemento. Quindi il programma chiede all'utente quali elementi hanno valore diverso da "1". L'utente fornirà gli elementi diversi da "1" dando per ciascuno una lista di tre elementi `[i,j,val]` in cui `i` é l'indice di riga, `j` é l'indice di colonna e `val` é il valore dell'elemento. La lettura continua fino a quando l'utente scrive `basta`.

```
?-  run(M, MT).

Dimensione ?
4.

Elementi diversi da 1?
[1, 1, 11].
[2, 1, 21].
[3, 2, 32].
[4, 3, 43].
[4, 2, 42].
basta.

M = [[11, 1, 1, 1], [21, 1, 1, 1], [1, 32, 1, 1], [1, 42, 43, 1]]
MT = [[11, 21, 1, 1], [1, 1, 32, 42], [1, 1, 1, 43], [1, 1, 1, 1]] ->;

no
```

**Soluzione 58.**

```
trasp([[]|_],[]).
trasp(Matrice,[Col1|ColN]):-
      colonne(Matrice,Col1,RestoColonne),
      trasp(RestoColonne,ColN).

colonne([],[],[]).
colonne([[C11|C1N]|C],[C11|X],[C1N|Y]):-
      colonne(C,X,Y).

run(MAT1,MATT):-
      read(N), nl,
      costr(N,MAT),
      read(EL),
      leggi(EL,MAT,MAT1),
      trasp(MAT1,MATT).

costr(N,MAT):-
      listadi1(N,L1),
      costr0(N,L1,MAT).

listadi1(0,[]):- !.
```

```
listadi1(N,[1|C]):-
      N1 is N - 1,
      listadi1(N1,C).

costr0(0,L1,[]):- !.
costr0(N,L1,[L1|C]):-
      N1 is N - 1,
      costr0(N1,L1,C).

leggi(basta,MATT,MATT):- !.
leggi(EL,MAT,MATT):-
      agg(EL,1,MAT,MAT1),
      read(EL1),
      leggi(EL1,MAT1,MATT).

agg([I,J,VAL],N,[RIGA|CMAT],[RIGA|CMAT1]):-
       I \= N,
       N1 is N + 1,
       agg([I,J,VAL],N1,CMAT,CMAT1).
agg([I,J,VAL],N,[RIGA|CMAT],[RIGA1|CMAT]):-
       I = N,
       agg0(J,VAL,1,RIGA,RIGA1).

agg0(J,VAL,N,[T|CRIGA],[T|CRIGA1]):-
       J \= N,
       N1 is N + 1,
       agg0(J,VAL,N1,CRIGA,CRIGA1).
agg0(J,VAL,N,[T|CRIGA],[T1|CRIGA]):-
       J = N,
       T1 = VAL.
```

**Esercizio 59.**

Si supponga di avere nel Data Base PROLOG un insieme di fatti a due argomenti `pezzi(CODICE,QUANTITA)`, dove `CODICE` è il codice numerico di un prodotto e `QUANTITA` è la corrispondente quantità di prodotto presente nel magazzino. Scrivere un predicato `run(TOTALE)` che legga da tastiera un codice-schema `CODS` (descritto sotto) e restituisca il totale dei pezzi disponibili corrispondenti presenti. Il codice-schema `CODS` permette di specificare gruppi di codici mediante un carattere jolly, consistente nel punto interrogativo (`?`) che sostituisce una singola cifra e può essere presente un numero di volte qualsiasi. Si supponga che il Data Base sia molto grande e che il numero di caratteri jolly nel `CODS` sia sempre piccolo. Ad esempio, si suppongano presenti nel Data Base i seguenti fatti

```
pezzi(23412,2). pezzi(23414,4). pezzi(23415,3). pezzi(23418,2).
pezzi(23512,1). pezzi(23514,2). pezzi(23518,2). pezzi(23519,8).
pezzi(23712,4). pezzi(23714,5). pezzi(23717,5). pezzi(23519,2).
pezzi(43412,1). pezzi(53412,2). pezzi(63412,9). pezzi(26412,1).
```

Si hanno le seguenti chiamate e risposte

```
?- run(TOTALE).
dai il codice-schema
'23?1?' .
TOTALE = 40 ->;

no


?- run(TOTALE).
dai il codice-schema
'?3?1?' .

TOTALE = 52 ->;

no
```

**Soluzione 59.**

```
alfabeto(['0','1','2','3','4','5','6','7','8','9']).

run(T):-
      write('dai il codice'), nl,
      read(X),
      atom_chars(X,XA),
      findall(Y,opera(XA,'?',Y),L),
      somma(L,T).

opera(XA,PD,Y):-
      opera0(XA,PD,TA),
      lista_num(TA,T),
      pezzi(T,Y).

opera0(XA,PD,XA):-
      \+ member(PD,XA),
      !.
opera0(XA,PD,TA):-
      member(PD,XA),
      !,
      cancella(XA,PD,PRIMA,DOPO),
      alfabeto(LETTERE),
      member(LETT,LETTERE),
      append(PRIMA,[LETT],L),
      opera0(DOPO,PD,DOPOR),
      append(L,DOPOR,TA).

cancella([T|C],T,[],C).
cancella([T|C],T1,[T|C1],C2):-
```

```
      T1 \= T,
cancella(C,T1,C1,C2).

lista_num(L,N):-
      car_int_lista(L,L1),
      reverse(L1,LR),
      lista_num0(LR,0,N,0).

car_int_lista([],[]):-
      !.
car_int_lista([T|C],[T1|C1]):-
      name(T,U),name(T1,U),
      car_int_lista(C,C1).

lista_num0([],_,ACC,ACC):-
      !.
lista_num0([T|C],CONT,N,ACC):-
      ACC1 is ACC + T*10^CONT,
      CONT1 is CONT+1,
      lista_num0(C,CONT1,N,ACC1).

somma([],0).
somma([T|C],N):-
      somma(C,M),
      N is M + T.

pezzi(23412,2).
pezzi(23512,1).
pezzi(23712,4).
pezzi(43412,1).
pezzi(23414,4).
pezzi(23514,2).
pezzi(23714,5).
pezzi(53412,2).
pezzi(23415,3).
pezzi(23418,2).
pezzi(23518,2).
pezzi(23519,8).
pezzi(23717,5).
pezzi(23519,2).
pezzi(63412,9).
pezzi(26412,1).
```

**Esercizio 60.**

Dato un insieme $S$, chiamiamo ***nogoods*** di $S$ alcuni suoi sottoinsiemi. Dato un insieme $S$ ed un insieme $N$ di nogoods di $S$, chiamiamo *pregood* di ($S$,$N$) ogni sottoinsieme di $S$ che non é superinsieme di alcun nogood in $N$. Dato un insieme $S$ ed un insieme $N$ di nogoods di $S$, definiamo ***good*** di ($S$,$N$) ogni pregood di ($S$,$N$) che non é sottoinsieme di alcun altro pregood di ($S$,$N$). Sia $G$ l'insieme di tutti i good di ($S$,$N$).
Una lista *corrisponde* ad un insieme se ne contiene tutti e soli gli elementi. Siano `S` una lista corrispondente ad $S$, `N` una lista di liste corrispondente ad $N$ e `G` una lista di liste corrispondente a $G$.
Scrivere un predicato PROLOG `goods(S,N,G)` che, date `S` ed `N` restituisca `G`.
Suggerimento: scrivere un predicato PROLOG `ss(S,SS)` che restituisca, per risoddisfacimento, tutti i sottoisiemi `SS` di `S`.

```
?-  goods([a, b, c, d], [[a, b]], G)
G = [[a, c, d], [b, c, d]]
yes.

?-  goods([a, b, c, d], [[a, b], [b, c]], G)
G = [[a, c, d], [b, d]]
yes.

?-  goods([a, b, c, d], [[a, b], [b, c], [c, d]], G)
G = [[a, c], [a, d], [b, d]]
yes.

?-  goods([a, b, c, d], [[a, b], [b, c], [c, d], [d, a]], G)
G = [[a, c], [b, d]]
yes.

?-  goods([a, b, c, d], [[a], [b], [c], [d]], G)
G = [[]]
yes.

?-  goods([a, b, c, d], [[a], [b], [c]], G)
G = [[d]]
yes.

?-  goods([a, b, c, d], [[a], [b]], G)
G = [[c, d]]
yes.

?-  goods([a, b, c, d], [[a]], G)
G = [[b, c, d]]
yes.

?-  goods([a, b, c, d], [], G)
G = [[a, b, c, d]]
yes.
```

**Soluzione 60.**

```
goods(KB,Nogoods,Goods) :-
      findall(SS,s(KB,Nogoods,SS),PreGoods),
      eliminasottoinsiemi(PreGoods,PreGoods,Goods).

% SS é sottoinsieme di KB e non e superinsieme di alcuna lista
% in Nogoods
s(KB,Nogoods,SS) :-
      ss(KB,SS),
      not contiene_sotto(Nogoods,SS).

ss([],[]).
ss([T|C],[T|C1]) :-
      ss(C,C1).
```

```
ss([T|C],C1) :-
      ss(C,C1).

% sotto(A,B) verifica che la lista A é sottoinsieme della lista B
sotto([],I).
sotto([T|C],I) :-
      member(T,I),
      sotto(C,I).

% contiene_sotto(A,B) verifica che la lista di liste A contiene una
% lista che é sottoinsieme della lista  B
contiene_sotto([T|C],SS) :-
      sotto(T,SS),!.
contiene_sotto([T|C],SS) :-
      contiene_sotto(C,SS).

contiene_super_proprio([T|C],SS) :-
      sotto(SS,T),
      not sotto(T,SS),!.
contiene_super_proprio([T|C],SS) :-
      contiene_super_proprio(C,SS).

eliminasottoinsiemi([],_,[]).
eliminasottoinsiemi([T|C],PreGoods,[T|C1]) :-
      not contiene_super_proprio(PreGoods,T),
      eliminasottoinsiemi(C,PreGoods,C1).
eliminasottoinsiemi([T|C],PreGoods,C1) :-
      contiene_super_proprio(PreGoods,T),
      eliminasottoinsiemi(C,PreGoods,C1).

member(X,[X|_]).
member(X,[Y|C]) :- member(X,C).
```

**Esercizio 61.**

Scrivere un predicato PROLOG `numeri/0` che stampi in ordine crescente tutti i numeri interi componibili secondo le informazioni contenute nel DATA BASE di predicati-fatto `posto_cifra/2` e `ripetizioni/2`:
`posto_cifra(POSIZIONE, LISTA)`
`POSIZIONE` indica la posizione della cifra nel numero, `LISTA` è la lista delle cifre che non devono comparire nella posizione `POSIZIONE`; il numero di questi fatti varia di volta in volta
`ripetizioni(CIFRA, N_VOLTE)`
`CIFRA` indica una particolare cifra, `N_VOLTE` è il numero di volte che la cifra `CIFRA` può comparire nel numero. Ad esempio, supponendo che nel DATA BASE ci siano i seguenti fatti

```
posto_cifra(1,[0,2,4,9]).
posto_cifra(2,[1,2,3,4,5,6,7]).
posto_cifra(3,[1,3,5,7,9]).
ripetizioni(0,2).
ripetizioni(1,2).
ripetizioni(2,1).
ripetizioni(3,2).
ripetizioni(4,1).
ripetizioni(5,2).
ripetizioni(6,1).
ripetizioni(7,2).
ripetizioni(8,0).
ripetizioni(9,3).
```

si deve avere quanto segue

```
?- numeri.
100
102
104
106
190
192
194
196
300
302
304
306
390
392
394
396
500
502
504
506
590
592
594
596
600
602
604
690
692
694
700
702
704
706
790
792
794
```

```
796

yes
```

**Esercizio 62.**

Sia dato un database di fatti del tipo `connesso(L1,L2,D)`, dove `L1` e `L2` sono due città collegate da una strada di lunghezza `D`, percorribile in entrambi i sensi. Scrivere un programma PROLOG
`vai(Partenza,Arrivo,Lunghezza)`
che date le città di `Partenza` e di `Arrivo`, restituisca la `Lunghezza` del percorso più breve fra le due.
Scrivere inoltre un programma PROLOG
`distanze`
che individui tutte le città presenti nel database e, calcolata la distanza minima per ogni coppia di città, scriva in output la matrice triangolare delle distanze minime così come riportato nell'esempio.
Per semplicità di formattazione si supponga ciascuna città rappresentata da un'unica lettera dell'alfabeto e si supponga tutte le distanze rappresentabili con interi da 1 a 99. Ad esempio, con il seguente database:

```
connesso(a,b,1).
connesso(b,c,2).
connesso(c,d,3).
connesso(c,e,4).
connesso(e,f,5).
connesso(f,b,6).
connesso(f,g,7).
connesso(i,f,8).
connesso(l,i,9).
connesso(l,h,10).
connesso(g,h,11).
connesso(h,m,12).

?- distanze.

a  1  3  6  7  7  14 25 15 24 37
   b  2  5  6  6  13 24 14 23 36
      c  3  4  8  15 26 16 25 38
         d  7  11 18 29 19 28 41
            e  5  12 23 13 22 35
               f  7  18 8  17 30
                  g  11 15 21 23
                     h  19 10 12
                        i  9  31
                           l  22
                              m
yes
```

**Soluzione 62.**

```
distanze :-
      setof(Citta,Y^D^conn(Citta,Y,D),ListaCitta),
      stampadistanze(ListaCitta,0).

conn(X,Y,D) :- connesso(X,Y,D).
conn(X,Y,D) :- connesso(Y,X,D).

stampadistanze([T],Ind) :- !, tab(Ind),writenl(T).
stampadistanze([T|C],Ind) :-
      tab(Ind),write(T),write('  '),
      stampa(T,C),
      Ind1 is Ind + 3,
      stampadistanze(C,Ind1).
stampa(T,[T1]) :- !,
      vai(T,T1,D),
      scrivi(D), nl.
stampa(T,[T1|C1]) :-
      vai(T,T1,D),
```

```
      scrivi(D),
      stampa(T,C1).

scrivi(X) :- X>=10,write(X),write(' ').
scrivi(X) :- X<10,write(X),write('  ').

vai(Partenza,Arrivo,Distanza) :-
      setof(Dist,tragitto(Partenza,Arrivo,[],0,Dist),[Distanza|_]).

tragitto(Arrivo,Arrivo,Passati,Dist,Dist).
tragitto(Partenza,Arrivo,Passati,Parziale,Dist) :-
      conn(Partenza,Verso,Distanza),
      not(member(Verso,Passati)),
      NuovoParziale is Parziale + Distanza,
      tragitto(Verso,Arrivo,[Partenza|Passati],NuovoParziale,Dist).

member(X,[X|_]).
member(X,[_|Z]) :- member(X,Z).
```

**Esercizio 63.**

Sia dato un database di fatti del tipo `costante(C)`, `variabile(V)`, `funzione(F,Arità)` e `predicato(P,Arità)`, dove `C`, `V`, `F` e `P` sono istanziabili ad atomi qualsiasi e `Arità` é istanziabile ad un intero qualsiasi. L'interpretazione ovvia é che detti atomi costituiscono l'alfabeto di un linguaggio del 1° ordine L. Scrivere un programma PROLOG `termine(X)` che prenda in `X` una struttura e termini con successo se essa rappresenta un termine di L. Rappresentando inoltre i connettivi logici `non`, `or`, `and` e `->` come operatori PROLOG (con le dovute precedenze) ed i quantificatori `all(V,F)` e `exists(V,F)` come funtori PROLOG, dove `V` ed `F` sono rispettivamente indice e campo d'azione del quantificatore, si scriva un programma PROLOG `fbf(X)` che prenda in `X` una struttura e termini con successo se essa rappresenta una formula ben formata di L. Ad esempio, con il seguente database si ottengono le risposte riportate di seguito:

```
costante(a).
costante(b).
costante(c).
costante(d).
funzione(f,1).
funzione(g,2).
funzione(h,3).
funzione(i,4).
predicato(p,1).
predicato(q,2).
predicato(r,3).
predicato(s,4).
variabile(x).
variabile(y).
variabile(z).

?-  fbf(all(x,p(x) -> q(a,g(x,b))))
yes

?-  fbf(exists(y,all(x,p(x) -> q(a,g(x,b)))))
yes

?-  fbf(r(a,b,c) or exists(y,all(x,p(x) -> q(a,g(x,b)))))
yes

?-  fbf(r(a,b,c,d) or exists(y,all(x,p(x) -> q(a,g(x,b)))))
no

?-  fbf(all(x,r(a,b,c)) or exists(y,all(x,p(x) -> q(a,g(x,b)))))
yes

?-  fbf(all(x,h(a,b,c)) or exists(y,all(x,p(x) -> q(a,g(x,b)))))
no

?-  fbf(h(a,b,c) or exists(y,all(x,p(x) -> q(a,g(x,b)))))
no

?-  fbf(r(a,b,c) or exists(y,all(x,p(x) -> q(a,g(x,b)))))
no
```

**Soluzione 63.**

```
 :-  op(30,fx,non).
 :-  op(100,yfx,or).
 :-  op(100,yfx,and).
 :-  op(150,yfx,->).
 :-  op(150,yfx,<->).

termine(X) :-
      costante(X).
termine(X) :-
      variabile(X).
termine(X) :-
```

```
      X =.. [T|C],
      functor(X,T,N),
      funzione(T,N),
      termini(C).

termini([]).
termini([T|C]) :-
      termine(T),
      termini(C).

fbf(X) :-
      X =.. [T|C],
      functor(X,T,N),
      predicato(T,N),
      termini(C).

fbf(all(V,F)) :- variabile(V), fbf(F).
fbf(exists(V,F)) :- variabile(V), fbf(F).
fbf(non X) :- fbf(X).
fbf(X or Y) :- fbf(X), fbf(Y).
fbf(X and Y) :-  fbf(X), fbf(Y).
fbf(X -> Y) :-  fbf(X), fbf(Y).
fbf(X <-> Y) :-  fbf(X), fbf(Y).
```

**Esercizio 64.**

Una *sostituzione* θ è un'insieme finito della forma $\{v_1/t_1,...,v_n/t_n\}$, dove ogni $v_i$ è una variabile, ogni $t_i$ è un termine distinto da $v_i$ e le variabili $v_1,...,v_n$ sono distinte. Una *espressione* è un termine oppure un atomo. Sia $\theta=\{v_1/t_1,...,v_n/t_n\}$ una sostituzione ed E un'espressione. Allora Eθ, l'*istanza* di E per mezzo di θ, è l'espressione ottenuta da E rimpiazzando ogni occorrenza della variabile $v_i$ in E con il termine $t_i$ (i=1,...,n).
Siano $\theta=\{u_1/s_1,...,u_m/s_m\}$ e $\sigma=\{v_1/t_1,...,v_n/t_n\}$ due sostituzioni. La *composizione* θσ di θ e σ è la sostituzione ottenuta dall'insieme $\{u_1/s_1\sigma,...,u_m/s_m\sigma, v_1/t_1,...,v_n/t_n\}$ eliminando ogni legame $u_i/s_i\sigma$ in cui $u_i=s_i\sigma$ ed eliminando ogni legame $v_j/t_j$ in cui $v_j\in\{u_1,...,u_m\}$.
Si rappresentino le espressioni come strutture PROLOG, gli elementi di una sostituzione come strutture `V/T` (dove `V` é un atomo, `T` é un atomo o una strutura) e le sostituzioni come liste di strutture `V/T`. Si supponga predefinito un predicato `substitute(V,T,FV,FT)` in cui `V` é un atomo, `T` é un atomo o una struttura, `FV` é una struttura e `FT` e' la struttura `FV` in cui ogni occorrenza dell'atomo `V` e' stata sostituita da `T`.
Si scrivano due predicati:

- `istanzia(Espressione,Sostituzione,Istanza)`

che data un'`Espressione` ed una `Sostituzione`, restituisca l'`Istanza` di `Espressione` mediante `Sostituzione`;

- `componi(Sostituzione1,Sostituzione2,Composizione)`

che date le due sostituzioni ne restituisca la corrispondente composizione.
Per semplicità, si supponga che, data una sostituzione `[V1/T1,...,Vn/Tn]`, nessuna delle variabili rappresentate dagli atomi `V1,...,Vn` compaia in alcuno dei termini rappresentati da `T1,...,Tn`.

```
?- istanzia(p(x,f(y),g(z)),[x/f(y),z/g(y)],Istanza).
Istanza = p(f(y), f(y), g(g(y)))->;
No

?- istanzia(p(x,f(y),g(z)),[y/f(b),z/b],Istanza).
Istanza = p(x, f(f(b)), g(b))->;
No

?- componi([y/f(b),z/b],[x/f(y),z/g(y)],Composizione).
Composizione = [y/f(b), z/b, x/f(y)]->;
No

?- componi([x/f(y),z/g(y)],[y/f(b),z/b],Composizione).
Composizione = [x/f(f(b)), z/g(f(b)), y/f(b)]->;
No
```

**Soluzione 64.**

```
istanzia(Espressione,[],Espressione).
istanzia(Espressione,[Variabile/Termine|C],Istanza) :-
      substitute(Variabile,Termine,Espressione,New),
      istanzia(New,C,Istanza).

componi(S1,S2,SostComp) :-
      componi1(S1,S2,S),
      variabili(S1,Variabili_S1),
      riduci(Variabili_S1,S2,S2ridotta),
      append(S,S2ridotta,S3),
      elimina(S3,SostComp).

componi1([],S2,[]).
componi1([V1/Term1|C1],S2,[V1/NewTerm1|C]) :-
      istanzia(Term1,S2,NewTerm1),
      componi1(C1,S2,C).

riduci(Variabili,[],[]).
```

```
riduci(Variabili,[V/T|C],[V/T|C1]) :-
      not member(V,Variabili),
      riduci(Variabili,C,C1).
riduci(Variabili,[V/T|C],C1) :-
      member(V,Variabili),
      riduci(Variabili,C,C1).

variabili([],[]).
variabili([V/T|C],[V|C1]) :-
      variabili(C,C1).

elimina([],[]).
elimina([V/T|C],[V/T|C1]) :-
      V\=T,
      elimina(C,C1).
elimina([V/T|C],C1) :-
      V=T,
      elimina(C,C1).

member(T,[T|C]).
member(T,[T1|C]) :- T\=T1,member(T,C).

substitute(Old,New,Old,New) :- !.
substitute(Old,New,Term,Term):-
       atomic(Term),
       Term \= Old.
substitute(Old,New,Term,Term1):-
       not(atomic(Term)),
       functor(Term,F,N),
       functor(Term1,F,N),
       substitute(N,Old,New,Term,Term1).

substitute(N,Old,New,Term,Term1):-
       N > 0,
       arg(N,Term,Arg),
       substitute(Old,New,Arg,Arg1),
       arg(N,Term1,Arg1),
       N1 is N - 1,
       substitute(N1,Old,New,Term,Term1).
substitute(0,Old,New,Term,Term1).
```

**Esercizio 65.**

Sia dato un dizionario rappresentato nel database PROLOG di fatti del tipo `p(`*`Parola`*`)`, dove *`Parola`* é un atomo corrispondente alla parola del dizionario. Scrivere un predicato PROLOG `cruciverba(M,N)` che usi le parole nel dizionario per costruire e stampare riga per riga un cruciverba tutto pieno di `M` righe ed `N` colonne (`M` e `N` sono instanziati alla chiamata). Righe e colonne sono occupate da un'unica parola (`M` parole di `N` lettere ed `N` parole di `M` lettere). Il programma fornisca, per risoddisfacimento, tutti i possibili cruciverba ricavabili dal dizionario assegnato.
Si discuta con brevi parole come si potrebbe estendere il programma per comprendere il caso normale in cui nel cruciverba compaiono caselle nere. Ad esempio, supponendo di avere il seguente database:

```
p(abaco).
...
p(aia).
...
p(amai).
...
p(car).
...
p(cera).
...
p(emo).
...
p(ram).
...
p(roma).
...
p(zuzzurellone).


?- cruciverba(3,4).

cera
amai
roma

-> ;
...
```

**Soluzione 65.**

```
cruciverba(M,N) :-
      cruciverba0([],L,M,N), scrivi_cruciverba(L).

scrivi_cruciverba([]).
scrivi_cruciverba([T|C]) :-
      name(P,T), writenl(P), scrivi_cruciverba(C).

cruciverba0(Istanziato,Istanziato,0,N) :-
      controlla(Istanziato,N).
cruciverba0(L,L1,M,N) :-
      M>0, M1 is M-1, inseribile(X,N), cruciverba0([X|L],L1,M1,N).

controlla(C,0).
controlla(C,N) :-
      N>0, verificaparola(C,N,[]), N1 is N-1, controlla(C,N1).

verificaparola([],N,P) :-
      reverse(P,Parola), name(Par,Parola), p(Par).
verificaparola([X|L],N,P) :-
      lettera(X,N,Lettera), verificaparola(L,N,[Lettera|P]).

lettera([X|_],1,X).
lettera([_|X],N,L) :-
      N1 is N-1, lettera(X,N1,L).
```

```
inseribile(L,M) :-
      parola(L), lung(M,L).

lung(0,[]).
lung(L,[T|C]) :-
      lung(LC,C), L is LC+1.

parola(Parola) :-
      p(P), name(P,Parola).

reverse(X,R) :- reverse(X,[],R).
reverse([T|C],Acc,R) :-
      reverse(C,[T|Acc],R).
reverse([],R,R).
```

**Esercizio 66.**

Sia dato un database PROLOG di fatti del tipo `pezzo(Id,L)`, dove *Id* é un atomo che identifica un pezzo di tavola lungo *L* centimetri.
Scrivere un predicato PROLOG `tavola(R)`, dove `R` é la lunghezza della tavola da cui ricavare i pezzi, che esegua quanto segue:

1. costruisca la lista di *tutti* gli elementi del tipo `(L,Id)` corrispondenti ai pezzi nel database, ordinata decrescentemente rispetto ad *L*
2. costruisca la lista di tutte le possibili liste del tipo precedente, tali che:

a) la somma `S` delle lunghezze dei pezzi non superi `R`
b) non rimanga fuori dalla lista un pezzo di luinghezza minore o uguale ad `R-S`

3. ordini crescentemente la lista di liste rispetto allo sfrido prodotto `R-S`
4. scriva in output tutte le combinazioni di pezzi otenibili tagliando una tavola lunga `R`, da quella che produce il minimo sfrido a quella che produce il massimo.

Suggerimento: usare `setof`. Ad esempio, supponendo di avere il seguente database:

```
pezzo(a,85).
pezzo(b,75).
pezzo(c,65).
pezzo(d,55).
pezzo(e,45).
pezzo(f,35).
pezzo(g,25).
pezzo(h,15).

?-  tavola(200)

0    [(15,h), (25,g), (75,b), (85,a)]
0    [(15,h), (35,f), (65,c), (85,a)]
0    [(15,h), (45,e), (55,d), (85,a)]
0    [(15,h), (45,e), (65,c), (75,b)]
0    [(25,g), (35,f), (55,d), (85,a)]
0    [(25,g), (35,f), (65,c), (75,b)]
0    [(25,g), (45,e), (55,d), (75,b)]
0    [(35,f), (45,e), (55,d), (65,c)]
5    [(15,h), (25,g), (35,f), (45,e), (75,b)]
5    [(15,h), (25,g), (35,f), (55,d), (65,c)]
5    [(35,f), (75,b), (85,a)]
5    [(45,e), (65,c), (85,a)]
5    [(55,d), (65,c), (75,b)]
10    [(15,h), (25,g), (65,c), (85,a)]
10    [(15,h), (35,f), (55,d), (85,a)]
10    [(15,h), (35,f), (65,c), (75,b)]
10    [(15,h), (45,e), (55,d), (75,b)]
10    [(25,g), (35,f), (45,e), (85,a)]
10    [(25,g), (35,f), (55,d), (75,b)]
10    [(25,g), (45,e), (55,d), (65,c)]
15    [(15,h), (25,g), (35,f), (45,e), (65,c)]
20    [(15,h), (25,g), (55,d), (85,a)]
20    [(15,h), (25,g), (65,c), (75,b)]
20    [(15,h), (35,f), (45,e), (85,a)]
20    [(15,h), (35,f), (55,d), (75,b)]
20    [(15,h), (45,e), (55,d), (65,c)]
25    [(15,h), (25,g), (35,f), (45,e), (55,d)]
30    [(15,h), (25,g), (45,e), (85,a)]
30    [(15,h), (25,g), (55,d), (75,b)]
40    [(15,h), (25,g), (35,f), (85,a)]
yes
```

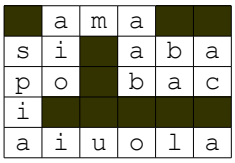
**Soluzione 66.**

```
tavola(Risorsa) :-
      setof((L,X),pezzo(X,L),P),
      reverse(P,Pezzi),
      setof((S,C),taglia(Pezzi,0,Risorsa,[],C,S),TagliPossibili),
```

```
      riduci(TagliPossibili,Tagli,TagliPossibili),
      scrivi(Tagli).

riduci([],[],_).
riduci([(S,C)|AltriPossibili],[(S,C)|Altri],TagliPossibili) :-
      not sotto(C,TagliPossibili),
      riduci(AltriPossibili,Altri,TagliPossibili).
riduci([(S,C)|AltriPossibili],Altri,TagliPossibili) :-
      riduci(AltriPossibili,Altri,TagliPossibili).

scrivi([]).
scrivi([(C,S)|Altri]) :-
      write(C),tab(4),writenl(S),scrivi(Altri).

taglia([(LX,X)|Altre],LP,LM,CP,C,Resto) :-
      LP1 is LP + LX,
      LP1 =< LM,
      taglia(Altre,LP1,LM,[(LX,X)|CP],C,Resto).
taglia([(LX,X)|Altre],LP,LM,CP,C,Resto) :-
      taglia(Altre,LP,LM,CP,C,Resto).
taglia([],LP,LM,C,C,Resto) :-
      Resto is LM - LP.

sotto(SL,[(S,L)|Altri_Possibili]) :- sottoinsieme(L,SL), SL \= L.
sotto(SL,[(S,_)|Altri_Possibili]) :- sotto(SL,Altri_Possibili).

sottoinsieme(_,[]).
sottoinsieme([T|C],[T|C1]) :-
      sottoinsieme(C,C1).
sottoinsieme([T|C],C1) :-
      sottoinsieme(C,C1).

reverse(X,R) :- reverse(X,[],R).
reverse([T|C],Acc,R) :-
      reverse(C,[T|Acc],R).
reverse([],R,R).
```

**Esercizio 67.**

Sia rappresenti un cruciverba come database di fatti `a(X,Y,Lettera)`, dove `Lettera` é un atomo che rappresenta una lettera dell'alfabeto.
Sia dato inoltre un database di fatti del tipo `p(Parola)`, dove `Parola` é un atomo che rappresenta una parola del dizionario.
Scrivere un programma Prolog `trovaparole/0`, che scriva in output tutte le stringhe massimali contenute nel cruciverba, sia in orizzontale che in verticale, che corrispondono a parole del dizionario. Le parole devono essere scritte ordinate in senso alfabetico. Esempio:




```
p(aiuola).
p(spia).
p(bo).
p(aio).
p(ama).
p(si).
p(no).
p(po).

?- trovaparole.

aio
aiuola
ama
po
si
spia

yes
```

**Soluzione 67.**

```
trovaparole :-
      cercaparola(Parola),
      write(Parola),
      nl,
      fail.
trovaparole.

cercaparola(P) :-
      (cercaparolaorizzontale(Parola);cercaparolaverticale(Parola)),
      parola(Parola,P).

cercaparolaorizzontale(Parola) :-
      a(X,Y,L),
      not sinistra(X,Y,L,Xs,Y,Ls),
      orizzontale(X,Y,L,[L],Parola).

cercaparolaverticale(Parola) :-
      a(X,Y,L),
      not sopra(X,Y,L,X,Ys,Ls),
      verticale(X,Y,L,[L],Parola).

orizzontale(X,Y,L,Parola,Parola) :-
      not destra(X,Y,L,_,_,_).
orizzontale(X,Y,L,Parziale,Parola) :-
      destra(X,Y,L,Xd,Y,Ld),
      append(Parziale,[Ld],NuovoParziale),
```

```
      orizzontale(Xd,Y,Ld,NuovoParziale,Parola).

verticale(X,Y,L,Parola,Parola) :-
      not sotto(X,Y,L,_,_,_).
verticale(X,Y,L,Parziale,Parola) :-
      sotto(X,Y,L,X,Ys,Ls),
      append(Parziale,[Ls],NuovoParziale),
      verticale(X,Ys,Ls,NuovoParziale,Parola).

sinistra(X,Y,L,Xs,Y,Ls) :-
      Xs is X-1,
      !,
      a(Xs,Y,Ls).

destra(X,Y,L,Xd,Y,Ld) :-
      Xd is X+1,
      !,
      a(Xd,Y,Ld).

sopra(X,Y,L,X,Ys,Ls) :-
      Ys is Y+1,
      !,
      a(X,Ys,Ls).

sotto(X,Y,L,X,Ys,Ls) :-
      Ys is Y-1,
      !,
      a(X,Ys,Ls).

parola(Parola,P) :-
      stringof(Parola,P),
      p(P).
```

**Esercizio 68.**

Una sequenza di caratteri è palindroma se la si può leggere anche in senso inverso. Si scriva un programma Prolog `ord_palindromie(+Input,+Output)` che scriva nel file letto in `Output` tutte le sequenze di caratteri palindrome contenute nel file di testo letto in `Input`, ordinate in base alla loro lunghezza. Si preprocessi il file di `Input` in maniera da mettere tutte le lettere in minuscolo ed eliminare i caratteri ASCII che non sono lettere.

| input.txt | output.txt |
|---|---|
| "Avida di vita,<br>desiai ogni amore vero,<br>ma ingoiai sedativi,<br>da diva qual'ero."<br><br>Marylin Monroe | lunghezza palindroma<br><br>3 dad<br>3 ivi<br>3 iai<br>3 eve<br>3 iai<br>3 ivi<br>3 dad<br>5 idadi<br>5 rever<br>5 idadi<br>7 vidadiv<br>7 orevero<br>7 vidadiv<br>9 moreverom<br>11 amoreveroma<br>13 iamoreveromai<br>15 niamoreveromain<br>17 gniamoreveromaing<br>19 ogniamoreveromaingo<br>21 iogniamoreveromaingoi<br>23 aiogniamoreveromaingoia<br>25 iaiogniamoreveromaingoiai<br>27 siaiogniamoreveromaingoiais<br>29 esiaiogniamoreveromaingoiaise<br>31 desiaiogniamoreveromaingoiaised<br>33 adesiaiogniamoreveromaingoiaiseda<br>35 tadesiaiogniamoreveromaingoiaisedat<br>37 itadesiaiogniamoreveromaingoiaisedati<br>39 vitadesiaiogniamoreveromaingoiaisedativ<br>41 ivitadesiaiogniamoreveromaingoiaisedativi<br>43 divitadesiaiogniamoreveromaingoiaisedativid<br>45 adivitadesiaiogniamoreveromaingoiaisedativida<br>47 dadivitadesiaiogniamoreveromaingoiaisedativid ad<br>49 idadivitadesiaiogniamoreveromaingoiaisedativid adi<br>51 vidadivitadesiaiogniamoreveromaingoiaisedativid adiv<br>53 avidadivitadesiaiogniamoreveromaingoiaisedativid adiva |

```
?- ord_palindromie('input.txt','output.txt').
true
```

**Soluzione 68.**

```
ord_palindromie(Input,Output):-
     forma_lista_caratteri(ListaFrase,Input),
     compatta(ListaFrase,LF),
     findall(X, palindroma(LF,X),L),
     quick(L,L1),
     tell(Output),
     write_int,nl,
     write_list(L1),
     told.

forma_lista_caratteri(Testo,Input) :-
      see(Input),
      leggi_lista_car(Testo),
      seen.
```

```
leggi_lista_car([C|R]) :-
      get0(C),
      C \== -1, !,
      leggi_lista_car(R).
leggi_lista_car([]).


compatta([],[]).
compatta([T|C],[T1|C1]) :-
     T > 64,
     T < 91,
     !,
     T1 is T + 32,
     compatta(C,C1).

compatta([T|C],[T|C1]) :-
     T > 96,
     T < 123,
     !,
     compatta(C,C1).
compatta([_|C],C1) :-
      compatta(C,C1).


palindroma(L1,S):-
     sottolista(S,L1),
     length(S,N),
     N>1,
     reverse(S,S).

sottolista(S,L):-
     append(P,_,L),
     append(_,S,P).

quick([],[]).
quick([N|C],L):-
      split(N,C,Lmin,Lmagg),
      quick(Lmin,Lmin_ord),
      quick(Lmagg,Lmagg_ord),
      append(Lmin_ord,[N|Lmagg_ord],L).

split(_,[],[],[]).
split(N,[N1|C],[N1|L1],L2):-
      length(N1,LN1),
      length(N,LN),
      LN1 =< LN,
      !,
      split(N,C,L1,L2).
split(N,[N1|C],L1,[N1|L2]):-
      split(N,C,L1,L2).

write_int:-
   write(lunghezza),tab(3),write(palindroma),nl.

write_list([]):- nl.
write_list([T|C]):-
  length(T,N),
  name(Parola,T),
  tab(4),write(N),tab(8),write(Parola),nl,
  write_list(C).
```

**Esercizio 69.**

Un insieme di matrimoni si dice “stabile” se *non* esistono nella comunità due persone che si attraggono più di quanto siano attratti dal proprio partner.

Sia dato un database di fatti del tipo:

```
/* pref(Sesso, Persona, Lista) e' vero se Persona gradisce le
/* persone di sesso opposto nell'ordine dato da Lista.

pref(m, al, [lia,meg,pia,mia,sue]).
pref(m, bob, [pia,lia,mia,sue,meg]).
pref(m, ugo, [lia,mia,meg,sue,pia]).
pref(m, tom, [pia,mia,lia,sue,meg]).
pref(m, ken, [meg,mia,lia,pia,sue]).

pref(f, pia, [ken,al,tom,bob,ugo]).
pref(f, lia, [tom,ken,bob,al,ugo]).
pref(f, mia, [al,tom,bob,ugo,ken]).
pref(f, sue, [ugo,bob,tom,al,ken]).
pref(f, meg, [tom,bob,ugo,ken,al]).
```

Si lavori ad un programma PROLOG

***stabili(-Matrimoni)***

che per risoddisfacimento generi tutte le possibili combinazioni di matrimoni stabili fra tutti i soggetti nel database. Es:

```
?- stabili(M).
M = [m(al, pia), m(bob, sue), m(ugo, meg), m(tom, mia), m(ken, lia)] ;
M = [m(al, pia), m(bob, meg), m(ugo, sue), m(tom, mia), m(ken, lia)] ;
M = [m(al, mia), m(bob, sue), m(ugo, meg), m(tom, lia), m(ken, pia)] ;
M = [m(al, mia), m(bob, meg), m(ugo, sue), m(tom, lia), m(ken, pia)] ;
false.
```

Il suggerimento è quello di adottare un approccio del tipo *genera* (tutte le soluzioni candidate) *e testa* (la validità di ciascuna secondo la specifica data).

Soluzione 1:

```
stabili(M) :-
      findall(X,pref(m,X,_),Uomini),
      findall(Y,pref(f,Y,_),Donne),
      genera_e_testa(Uomini,Donne,M).

/* genera_e_testa(Uomini,Donne,M) è vero se M è un insieme di */
/*  matrimoni stabili fra Uomini e Donne                      */

genera_e_testa(Uomini, Donne, Matrimoni):-
  genera(Uomini, Donne, Matrimoni),
  stabile(Uomini, Donne, Matrimoni).

/* genera(Uomini, Donne, M) è vero se M è un insieme di matrimoni*/
/* possibili fra Uomini e Donne.                      */

genera([], [], []).
genera([Uomo|Uomini], Donne, [m(Uomo,Donna)|Matrimoni]):-
  select(Donna, Donne, Donne1),
  genera(Uomini, Donne1, Matrimoni).

/* stabile(Uomini, Donne, Matrimoni) è vero se Matrimoni è un insieme di */
/* matrimoni stabili fra Uomini e Donne.                                  */

stabile([], _, _).
stabile([Uomo|Uomini], Donne, Matrimoni):-
  stabile_1(Donne, Uomo, Matrimoni),
  stabile(Uomini, Donne, Matrimoni).

stabile_1([], _, _).
stabile_1([Donna|Donne], Uomo, Matrimoni):-
  not(instabile(Uomo, Donna, Matrimoni)),
  stabile_1(Donne, Uomo, Matrimoni).

/* instabile(Uomo, Donna, Matrimoni) è verso se Uomo e Donna si piacciono */
/* più del rispettivo partner come listato in Matrimoni.*/
instabile(Uomo, Donna, Matrimoni):-
  sposato(Uomo, Moglie, Matrimoni),
  sposato(Marito, Donna, Matrimoni),
  preferisce(Uomo, Donna, Moglie),
  preferisce(Donna, Uomo, Marito).

/* sposato(Uomo, Donna, Matrimoni) è verso se Uomo e Donna  sono     */
/* sposati come listato in Matrimoni.                          */

sposato(Uomo, Donna, Matrimoni):-
  resto(m(Uomo, Donna), Matrimoni, _).

/* preferisce(Persona, AltraPersona, Coniuge) è vero se Persona preferisce*/
/* AltraPersona al suo Coniuge.                                           */

preferisce(Persona, AltraPersona, Coniuge):-
  pref(_, Persona, Preferenze),
  resto(AltraPersona, Preferenze, Resto),
  resto(Coniuge, Resto, _).

/* resto(X, Ys, Zs) è vero se X appartiene alla lista Ys, e la lista   */
/*   Zs è il resto della lista dopo X.                                 */

resto(X, [X|Ys], Ys):- !.
resto(X, [_|Ys], Zs):- resto(X, Ys, Zs).

/* select(X, Ys, Zs) è vero se Zs è il risultato della rimozione di una   */
/* occorrenza dell'elemento X dalla lista Ys.                             */
select(X, [X|Ys], Ys).
```

```
select(X, [Y|Ys], [Y|Zs]):-select(X, Ys, Zs).
```

Soluzione 2:

```
stabili(M) :-
      findall(X,pref(m,X,_),Uomini),
      findall(Y,pref(f,Y,_),Donne),
      backtrack(Uomini,Donne,M).

backtrack(Uomini, Donne, Matrimoni):-
  backtrack_1(Uomini, Donne, Uomini, Donne, [], Matrimoni).

backtrack_1([], _, _, _, Matrimoni, Matrimoni).
backtrack_1([Uomo|Uomini], Donne, Uomini0, Donne0, Matrimoni0, Matrimoni):-
  select(Donna, Donne, Donne1),
  Matrimoni1 = [m(Uomo,Donna)|Matrimoni0],
  stabile(Uomini0, Donne0, Matrimoni1),
  backtrack_1(Uomini, Donne1, Uomini0, Donne0, Matrimoni1, Matrimoni).
```

**Esercizio 70.**

Due porzioni di DATABASE Prolog prima e seconda, di lunghezza non nota, contengono i risultati di una gara che si svolge in due manche. Ogni fatto della prima porzione contiene il nome del concorrente e un intero che rappresenta il punteggio ottenuto nella prima manche. Analogamente, ogni fatto della seconda porzione contiene il nome del concorrente e un intero che rappresenta il punteggio ottenuto nella seconda manche. Nella seconda porzione i fatti sono in sequenza diversa e i concorrenti che non sono presenti nella seconda porzione sono da ritenersi squalificati.
Scrivere un predicato Prolog `gara(L1,L2)` che restituisca:
1. nella lista `L1` i punteggi complessivi per ogni concorrente, strutturati in funtori concorrente/punteggio, ordinata in ordine decrescente rispetto al punteggio (a parità di punteggio non ha importanza l'ordine dei nomi)
2. nella lista `L2` i concorrenti squalificati in ordine alfabeticamente decrescente
esempio:

```
prima(sandro,15).
prima(amelia,9).
prima(marco,9).
prima(laura,6).
prima(paolo,7).
prima(sara,10).
prima(gigi,5).


seconda(sandro,10).
seconda(laura,6).
seconda(paolo,9).
seconda(marco,9).
seconda(amelia,9).
```

```
?- gara(L1,L2).

L1 = [sandro/25, amelia/18, marco/18, paolo/16, laura/12],

L2 = [gigi, sara] ;

false.
```

Soluzione:

```
gara(L1, L2) :-
        setof(Ptot/C, P1^P2^(prima(C,P1), seconda(C, P2), Ptot is P1+P2), S1),
        reord(S1, L1),
        % concorrenti squalificati ordinati in maniera alfabeticamente crescente
        setof(X, P1^P2^(prima(X,P1), \+ seconda(X,P2)), L2).

reord([], []).
reord([P/C|T], L) :-
           reord(T, T2),
           append(T2, [C/P], L).
```

**Esercizio 71.**

Si scriva un predicato PROLOG **`conta(L, MAX)`** dove **`L`** è istanziata ad lista di cifre. Trovare qual è la sequenza di ***due*** cifre che si ripete più volte, renderla come termine del tipo: **`CIFRA1/CIFRA2/RIPETIZIONI`** e restituirla nella *lista* **`MAX.`**
**`MAX`** è una lista perché ci possono essere delle sequenze di pari occorrenza massima, ed in tal caso la lista deve contenerle tutte.

**`Se L = [1,2,9,5,1,5,3,3,8,4,3,0,4,2,5,4,1,5,7,1]`**
**`MAX = [1/5/2]`**

**`Se L = [1,2,9,5,1,5,3,3,8,4,1,0,4,2,5,4,1,5,7,1]`**
**`MAX = [1/5/2 , 4/1/2]`**

**`Se L = [1,2,9,5,7,2,9,3,8,4,1,2,2,9,5,4,9,5,1,2]`**
**`MAX = [2/9/3, 9/5/3, 1/2/3]`**

Soluzione:

```
conta(L, Max) :-
         coppie(L, Coppie),
         setof(X/C, Y^Cy^(occorrenze(X, Coppie, C), \+ (occorrenze(Y, Coppie, Cy), Cy > C)),
Max).

coppie([_], []).
coppie([H,H1|T], [H/H1|T2]) :-
         coppie([H1|T], T2).

occorrenze(X, L, C) :-
         member(X, L),
         findall(_, member(X, L), Xs), length(Xs, C).
```

**Esercizio 72.**

Scrivere un predicato **istogramma**/1 che prende come argomento una lista di cifre (tra 0 e 9) e stampa un istogramma orizzontale rappresentante il numero di occorrenze di ciascuna, come nell'esempio.

```
?- istogramma([1,7,2,9,6,7,1,3,7,5,7,9]).
0
1 **
2 *
3 *
4
5 *
6 *
7 ****
8
9 **
true .
```

**SOLUZIONE**

```
istogramma(Numeri):-
     conta(Numeri,L),
     disegna(0,L).

conta(Numeri,L):-
     conta(Numeri,[],L).

disegna(N,_):-
     N > 9,
     !.
disegna(N,L):-
     N =< 9,
     write(N),tab(1),
     isto(N,L),
     nl,
     N1 is N+1,
     disegna(N1,L).

isto(N,L):-
     member(N/M,L),
     !,
     m_aster(M).
isto(N,L):-
     \+member(N/_,L),
     !.

m_aster(M):-
     M<1,
     !.
m_aster(M):-
     write('*'),
     M1 is M-1,
     m_aster(M1).

conta([],L,L).
conta([X|C],LPARZ,L):-
    contaX(X,C,1,NX),
    LPARZ1 = [X/NX|LPARZ],
    cancella_tutti(C,X,C1),
    conta(C1,LPARZ1,L).

contaX(_,[],NX,NX).
contaX(X,[X|C],NPARZ,NX):-
     NPARZ1 is  NPARZ +1,
     contaX(X,C,NPARZ1,NX).
contaX(X,[Y|C],NPARZ,NX):-
```

```
     X \= Y,
     contaX(X,C,NPARZ,NX).

cancella_tutti([],_,[]).
cancella_tutti([X|C],X,C1):-
     cancella_tutti(C,X,C1).
cancella_tutti([X|C],Y,[X|C1]):-
     X \= Y,
     cancella_tutti(C,Y,C1).
```

**Esercizio 73.**

Nel DB Prolog sono presenti tre fatti **a/1**, **b/1** e **c/1** i cui argomenti sono rispettivamente una lista L1 di interi, una lista L2 di interi e una lista LOP di atomi **+** , **-** e ***** . Scrivere un predicato **go/0** che valuti l'operazione di ogni elemento della lista LOP tra i corrispondenti interi di L1 e L2, come nell'esempio sotto riportato. Supponendo che nel DB Prolog siano presenti i seguenti fatti:

```
a([1,2,3,4]).
b([2,9,6,1]).
c(['+','-','*','+']).
```

si ha:

```
?- go.
[1+2,2-9,3*6,4+1]  % questa riga di stampa non è obbligatoria
[3,-7,18,5]
true.
```

**SOLUZIONE**

```
a([1,2,3,4]).
b([2,9,6,1]).
c(['+','-','*','+']).

go:-
      a(A),
      b(B),
      c(C),
      costruisci(A,B,C,L),
      write(L),nl,
      calcola(L,N),
      write(N).

costruisci([],[],[],[]).
costruisci([TA|CA],[TB|CB],[TC|CC],[T|C]):-
      T =.. [TC,TA,TB],
      costruisci(CA,CB,CC,C).

calcola([],[]).
calcola([T|C],[TN|CN]):-
      eval(T,TN),
      calcola(C,CN).

eval(X+Y,Z):-
      Z is X + Y.
eval(X-Y,Z):-
      Z is X - Y.
eval(X*Y,Z):-
      Z is X * Y.
```

**Esercizio 74.**

Scrivere un predicato Prolog **`ord_palindroma(L1)`**, dove L1 è una lista di interi, che stampi tutte le sue sottoliste palindrome (lunghezza > 1), in ordine crescente di lunghezza, come nell'esempio sotto riportato.

```
?- ord_palindroma([1,2,1,1,2,3,4,5,6,3,2,1,2,2,2,1]).
lunghezza   palindroma
    2        [2,2]
    2        [2,2]
    2        [1,1]
    3        [2,2,2]
    3        [2,1,2]
    3        [1,2,1]
    4        [2,1,1,2]
    5        [1,2,2,2,1]
true;
false
```

**SOLUZIONE**

```
pal(L1):-
      reverse(L1,L1).

sottolista(S,L):-
     append(P,_,L),
     append(_,S,P).

palindroma(L1,S):-
     sottolista(S,L1),
     length(S,N),
     N>1,
     pal(S).

ord_palindroma(L1):-
     findall(X, palindroma(L1,X),L),
     quick(L,L2),
     write_int,
     write_list(L2).

quick([],[]).
quick([N|C],L):-
      split(N,C,Lmin,Lmagg),
      quick(Lmin,Lmin_ord),
      quick(Lmagg,Lmagg_ord),
      append(Lmin_ord,[N|Lmagg_ord],L).

split(_,[],[],[]).
split(N,[N1|C],[N1|L1],L2):-
      length(N1,LN1),
      length(N,LN),
      LN1 =< LN,
      !,
      split(N,C,L1,L2).
split(N,[N1|C],L1,[N1|L2]):-
      split(N,C,L1,L2).

write_int:-
      write(lunghezza),tab(3),write(palindroma),nl.

write_list([]):- nl.
write_list([T|C]):-
      length(T,N),
      tab(4),write(N),tab(8),write(T),nl,
      write_list(C).
```

**Esercizio 75.**

Nel Data Base esiste un fatto Prolog **`cifre(L)`**, dove L è una lista di 4 interi a una cifra, non necessariamente in ordine. Scrivere un programma Prolog **`max(X,Y)`** che istanzi X e Y rispettivamente al più piccolo e al più grande numero che si può formare con le cifre contenute in L.
Se nel Data Base il fatto contenuto fosse **`cifre([0,3,9,6])`** si avrebbe:

```
?- max(X,Y).
X = 369,
Y = 9630.
```

SOLUZIONE

```
cifre([0,3,9,6]).

max(X,Y):-
     setof(N,max0(N),L),
     [X|_] = L,
     reverse(L,[Y|_]).

max0(N):-
     cifre(L),
     select(Prima,L,L1),
     select(Seconda,L1,L2),
     select(Terza,L2,L3),
     select(Quarta,L3,_),
     N is Prima * 1000+
          Seconda * 100 +
          Terza * 10 +
          Quarta * 1.
```

**Esercizio 76.**

Una sequenza di elementi su cui è definita una relazione d'ordine viene detta unimodale se consiste di una sottosequenza ascendente seguita da una sottosequenza discendente, presentando così un singolo picco. Sono compresi i casi particolari in cui una delle due sottosequenze, od entrambe, sono vuote o di un solo elemento. Scrivere un predicato che risulta vero per una lista unimodale di interi.

```
?- unimodale([1,2,4,9,8,3,0]).
true
?- unimodale([1,2,4,9]).
true

?- unimodale([1,2,4,9,8,3,0,1]).
false.
```

**SOLUZIONE**

```
unimodale(Lista) :-
     append(L1,L2, Lista),
     ascendente(L1),
     discendente(L2).

ascendente([]).
ascendente([_]).
ascendente([X,Y | Z]) :-
      X<Y,
      ascendente([Y|Z]).

discendente([]).
discendente([_]).
discendente([X,Y | Z]) :-
      X>Y,
      discendente([Y | Z]).
```

**Esercizio 77.**

Tre amici arrivano primo, secondo e terzo in una gara. Ciascuno ha un nome diverso, ama uno sport diverso e proviene da una nazione diversa. ***Richard*** ama la pallacanestro e si è piazzato meglio dell'americano. ***Michele***, l'israeliano, ha fatto meglio del giocatore di tennis. Il giocatore di cricket si è qualificato al primo posto.
*Domanda1.* Chi è l'australiano?
*Domanda2.* Quale sport pratica ***Simone***?

Scrivere un programma PROLOG `risolvi(Indizi,Domande,Soluzione)` per risolvere questo puzzle logico, in maniera da ottenere un output di questo tipo:

```
?- testa_puzzle(S).
S = [['Australiano é ', richard], ['Simone gioca a ', tennis]] ;
```

**SOLUZIONE**

```
risolvi(puzzle(Indizi,Domande,Soluzione),Soluzione) :-
    ris(Indizi),
    ris(Domande).

ris([Indizio|Indizi]) :-
    call(Indizio),
    ris(Indizi).

ris([]).

/* Test data */

  testa_puzzle(S) :-
    indizi([amico(N1,C1,S1), amico(N2,C2,S2), amico(N3,C3,S3)],Indizi),
    domande([amico(N1,C1,S1), amico(N2,C2,S2), amico(N3,C3,S3)],Domande,S),
    risolvi(puzzle(Indizi,Domande,S),S).

  indizi(Amici,
    [(fa_meglio(Am1Ind1, Am2Ind1, Amici),    % Indizio 1
          name_(Am1Ind1, richard),
          sport(Am1Ind1, basket),
          nationalita(Am2Ind1,americano)),
     (fa_meglio(Am1Ind2, Am2Ind2, Amici),    % Indizio 2
          name_(Am1Ind2, michele),
          nationalita(Am1Ind2,israeliano),
          sport(Am2Ind2, tennis)),
     (vincitore(Amici,AmInd3),                  % Indizio 3
          sport(AmInd3, cricket))
    ]).

  domande(Amici,
      [ member(Q1, Amici),
          nome(Q1, Nome),
          nationalita(Q1, australiano),        % Domanda 1
          member(Q2, Amici),
          nome(Q2, simone),
          sport(Q2, Sport)                     % Domanda 2
    ],
    [['Australiano è ', Nome], ['Simone gioca a ', Sport]]
  ).

  fa_meglio(A,B,[A,B,C]).
  fa_meglio(A,C,[A,B,C]).
  fa_meglio(B,C,[A,B,C]).

  nome(amico(A,B,C),A).
  nationalita(amico(A,B,C),B).
  sport(amico(A,B,C),C).

  vincitore([X|Xs],X).
```

**Esercizio 78**.

Ci sono due brocche non graduate aventi capacità di `n` ed `m` litri; il problema consiste nel prelevare esattamente `k` litri da un tino di capacità illimitata. Le operazioni consentite sono:

1. riempire una brocca prelevando dal tino
2. vuotare una brocca nel tino e trasferire i contenuti di una brocca nell'altra fino a che la brocca da cui si versa sia completamente vuota oppure sino a quando l'altra brocca sia completamente piena.

Risolvere il problema adottando una ricerca breadth first con arresto alla prima soluzione. Il predicato da realizzare è `go/0` e la stampa deve essere quella sottoriportata, che si riferisce al caso in cui le capacità dei due vasi sono 4 e 3 litri, e la quantità da isolare è di 2 litri.

```
?- go.

NodoIniz          brocche(0, 0)
riempi(2)         brocche(0, 3)
versa(2, 1)    brocche(3, 0)
riempi(2)         brocche(3, 3)
versa(2, 1)    brocche(4, 2)

Yes
```

SOLUZIONE

```
go:-
   start(Start),
   breadth_first_search(Start, Sol1),
   reverse(Sol1,Sol),
   write_list(Sol).

breadth_first_search(Start, Sol) :-
          bfs([['StartNode'-Start]], [ ], Sol).

bfs([[Mossa-Nodo|Path]|_],_,[Mossa-Nodo|Path]) :-
          goal(Nodo),
          !.
bfs([[Mossa-Nodo|Path]|MoreOPEN], CLOSED, Sol) :-
% trova i vicini del primo nodo OPEN ed aggiunge il percorso per ciascuno
findall(
      [Move-Next,Mossa-Node|Path],
      (
      (move(Node,Move),update(Node,Move,Next)),
      \+ member([_-Next|_], [[Mossa-Node|Path]|MoreOPEN]),
      \+ member(_-Next, CLOSED)
      ),
      NewPaths
      ),
      append(MoreOPEN, NewPaths, NewOPEN),  % CODA
      bfs(NewOPEN, [Mossa-Node|CLOSED], Sol).

goal(brocche(2,_)).
goal(brocche(_,2)).

start(brocche(0,0)).

capacita(1,4).
capacita(2,3).


move(brocche(V1,V2),riempi(1)):- capacita(1,C1),V1<C1.
move(brocche(V1,V2),riempi(2)):- capacita(2,C2),V2<C2.
move(brocche(V1,V2),vuota(1)):- V1 > 0.
move(brocche(V1,V2),vuota(2)):- V2 > 0.
move(brocche(V1,V2),versa(2,1)):- V2>0 .
move(brocche(V1,V2),versa(1,2)):- V1>0.

update(brocche(V1,V2),riempi(1),brocche(C1,V2)) :- capacita(1,C1).
```

```
update(brocche(V1,V2),riempi(2),brocche(V1,C2)) :- capacita(2,C2).
update(brocche(V1,V2),vuota(1),brocche(0,V2)).
update(brocche(V1,V2),vuota(2),brocche(V1,0)).
update(brocche(V1,V2),versa(2,1),brocche(W1,W2)) :-
  capacita(1,C1),
  Liquido is V1 + V2,
  Eccesso is Liquido - C1,
  aggiusta(Liquido,Eccesso,W1,W2).
update(brocche(V1,V2),versa(1,2),brocche(W1,W2)) :-
  capacita(2,C2),
  Liquido is V1 + V2,
  Eccesso is Liquido - C2,
  aggiusta(Liquido,Eccesso,W2,W1).

aggiusta(Liquido, Eccesso,Liquid,0) :-
  Eccesso =< 0.
aggiusta(Liquido,Eccesso,V,Eccesso) :-
  Eccesso > 0,
  V is Liquido - Eccesso.

write_list(L):-
  nl,write('Mossa'),
  tab(10),
  write('Stato'),nl,nl,
  write_list0(L).

write_list0([]):- nl.
write_list0([T|C]):-
  T = Mossa-Stato,
  write(Mossa),
  tab(5),
  write(Stato),nl,
  write_list0(C)
```

**Esercizio 79.**

Un sistema di gestione delle presenze sul luogo lavorativo assegna ad ogni impiegato un codice (un intero). La corrispondenza con il nome è contenuto nel file **`nomi.txt`**, in cui ogni riga è un fatto PROLOG
**`n(<INTERO>,<NOME>).`**
Ogni impiegato all'ingresso nel luogo di lavoro "timbra il cartellino", e così ad ogni uscita. Il sistema tiene traccia in due liste differenti le timbrature in ingresso e le timbrature in uscita: ogni volta che un impiegato entra/esce, il sistema aggiunge il codice identificativo in testa alla lista opportuna; uno stesso codice compare quindi ripetuto più volte. In un mondo perfetto il numero di ingressi sul luogo di lavoro ed il numero di uscite dovrebbe essere uguale. Invece ogni giorno ci sono dei problemi.

Il file **`codici.txt`** contiene tutti i codici degli impiegati
Il file **`entrate.txt`** contiene i codici degli impiegati che nella giornata odierna hanno timbrato l'ingresso (ad ogni timbro corrisponde un codice: più timbri di ingresso effettuati dalla stessa persona corrispondono allo stesso codice che compare più volte ripetuto nella lista);
Il file **`uscite.txt`** contiene i codici degli impiegati che nella giornata odierna hanno timbrato l'uscita (ad ogni timbro corrisponde un codice: più timbri di uscita effettuati dalla stessa persona corrispondono allo stesso codice che compare più volte ripetuto nella lista);
Tutti i 3 file sono costituiti da una serie di interi seguiti dal punto (uno per riga-vedi esempio).
Scrivere un predicato PROLOG **`go/0`** che stampi l'elenco degli impiegati assenti, di quelli con entrate e uscite regolari (il numero di uscite è uguale al numero di entrate) e di quelli non regolari.

*Esempio*

```
File nomi.txt
n(8,bianchi).
n(9,bruni).
n(3,caio).
n(7,sempronio)

File codici.txt
3.
9.
7.
8.

File entrate.txt
7.
8.
7.
9.
7.
9.

File uscite.txt
 7.
 9.
 9.
 9.
 7.
 7.

?- go.
NON VENUTI
caio

REGOLARI
sempronio

IRREGOLARI
bruni
bianchi

true
.
```

SOLUZIONE

```
go:-
    leggi(L1,L2,L3,NOMI),
    non_venuti(L1,L2,L3,NONVENUTI,L11),
    regolari(L11,L2,L3,REGOLARI,L111),
    irregolari(L111,L2,L3,IRREGOLARI,L1111),
    write('NON VENUTI '),nl,
    stampa(NONVENUTI,NOMI),
    write('REGOLARI '),nl,
    stampa(REGOLARI,NOMI),
    write('IRREGOLARI '),nl,
    stampa(IRREGOLARI,NOMI).

leggi(L1,L2,L3,NOMI):-
    leggi0(L1,'C:/Prolog-FILES/PS/prox/54/codici.txt'),
    leggi0(L2,'C:/Prolog-FILES/PS/prox/54/entrate.txt'),
    leggi0(L3,'C:/Prolog-FILES/PS/prox/54/uscite.txt'),
    leggi0(NOMI,'C:/Prolog-FILES/PS/prox/54/nomi.txt'),
    !.

leggi0(L,FILE):-
    open(FILE, read, Str),
    read_file(Str,L),
    close(Str).

read_file(Stream,[]) :-
    at_end_of_stream(Stream).

read_file(Stream,[X|L]) :-
    \+ at_end_of_stream(Stream),
    read(Stream,X),
    read_file(Stream,L).

non_venuti([],_,_,[],[]):-
    !.
non_venuti([T|C],L2,L3,[T|C1],L11):-
    \+ member(T,L2),
    \+ member(T,L3),
    !,
    non_venuti(C,L2,L3,C1,L11).
non_venuti([T|C],L2,L3,NONVENUTI,[T|C1]):-
    !,
    non_venuti(C,L2,L3,NONVENUTI,C1).

regolari([],_,_,[],[]):-
    !.
regolari([T|C],L2,L3,[T|C1],L111):-
    conta(T,L2,N2),
    conta(T,L3,N3),
    N2 == N3,
    !,
    regolari(C,L2,L3,C1,L111).
regolari([T|C],L2,L3,REGOLARI,[T|C1]):-
    !,
    regolari(C,L2,L3,REGOLARI,C1).

irregolari([],_,_,[],[]):-
    !.
irregolari([T|C],L2,L3,[T|C1],L111):-
    conta(T,L2,N2),
    conta(T,L3,N3),
    N2 \= N3,
    !,
    irregolari(C,L2,L3,C1,L111).
irregolari([T|C],L2,L3,REGOLARI,[T|C1]):-
    !,
    irregolari(C,L2,L3,REGOLARI,C1).
```

```
conta(_,[],0):-
    !.
conta(A,[A|C],N):-
    conta(A,C,N1),
    N is N1+1.
conta(A,[T|C],N):-
    T \=A,
    conta(A,C,N).

stampa([],_):-
    nl,!.
stampa([T|C],NOMI):-
    T1=n(T,NOME),
    member(T1,NOMI),
    write(NOME),nl,
    stampa(C,NOMI).
```

**Esercizio 80.**

Scrivere un predicato **terne/1** che unifichi in backtracking il suo argomento con le triple di una cifra [X,Y,Z], che soddisfino le seguenti condizioni

1) X, Y e Z siano tutte diverse tra loro
2) X e Z siano non nulle
3) (10X + Y)*Z = (10Y +Z)*X

Utilizzare il predicato **terne/1** per scrivere un predicato **go/0**, che stampi le triple e la somma totale.

```
?- go.
Le terne sono:
[4, 9, 8]
[2, 6, 5]
[1, 9, 5]
[1, 6, 4]


La somma è 60
```

SOLUZIONE

```
go:-
    findall([X,Y,Z],triple([X,Y,Z]),L), % mette il risultato in una lista di
                                        % triple
    calcola_somma(L,0,N),  % calcola la somma di tutte le triple
    stampa(L),  %stampa riga per riga le triple
    nl,write('La somma è '),write(N).  %stampa la somma totale

cifre([0,1,2,3,4,5,6,7,8,9]).

triple([X,Y,Z]):-
   cifre(Cifre), % Cifre contiene la lista dei numeri di 1 cifra
   member(X,Cifre),  % genera il primo numero
   member(Y,Cifre),  %           secondo
   member(Z,Cifre),  %           terzo
   X>0,Z>0,    % condizioni iniziali
   X \==Y,
   X \==Z,
   Y \== Z,
   (10*X+Y)*Z =:= (10*Y+Z)*X. % vincolo (test).

calcola_somma([],N,N).
calcola_somma([T|C],ACC,N):-
   calcola_somma0(T,0,N1),
   ACC1 is ACC+N1,
   calcola_somma(C,ACC1,N).

calcola_somma0([],N,N).
calcola_somma0([T|C],ACC,N):-
   ACC1 is ACC+T,
   calcola_somma0(C,ACC1,N).

stampa(L):-
   write('Le terne sono:'), nl,
   write_list(L).

write_list([]):- nl.
write_list([T|C]):-
  write(T),nl,
  write_list(C).


%cifre(X):- X=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15].
```

**Esercizio 81.**

Scrivere un predicato `encode1(L1,L2)`, che implementa il cosidetto *run-length-code*. Duplicati consecutivi di un elemento sono codificati come termini `[N,E]`, dove `N` è il numero di duplicati dell'elemento `E`. L'implementazione deve essere tale che se un elemento non ha duplicati questo è semplicemente copiato. Solo elementi con duplicato sono trasferiti come termini `[N,E]`

```
?- encode1([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).
X = [[4, a], b, [2, c], [2, a], d, [4, e]] ;
false.
```

SOLUZIONE

```
encode1(L1,L3) :-
     encode(L1,L2,[]),
     reverse(L2,L3).

%implementazione in tail recursion e deterministica
encode([],Acc,Acc).
encode([X],Ris,[[N,X]|C]) :-!,
     M is N+1,
     encode([],Ris,[[M,X]|C]).
encode([X],Ris,[X|C]) :-!,
     encode([],Ris,[[2,X]|C]).
encode([X],Ris,C) :-!,
     encode([],Ris,[X|C]).
encode([X|C],Ris,[[N,X]|C1]) :-!,
     M is N+1,encode(C,Ris,[[M,X]|C1]).
encode([X|C],Ris,[X|C1]) :-!,
     encode(C,Ris,[[2,X]|C1]).
encode([X|C],Ris,C1) :-
     encode(C,Ris,[X|C1]).
```

**Esercizio 82.**

Scrivere un predicato `accorcia(ATOMO1,ATOMO2)`, dove l'atomo `ATOMO2` e ottenuto dall'atomo `ATOMO1` in cui le lettere ripetute consecutive vengono rimpiazzate da una sola copia della lettera.

```
?- accorcia(annaaamammmmmmaa,ATOMO2).
ATOMO2 = anamama ;
false.

SOLUZIONE

accorcia(A,B):-
               name(A,L),
               elimina(L,L1),
               name(B,L1).

elimina([],[]):-!.
elimina([X],[X]):-!.
elimina([T,T|C],C1):- !,
     elimina([T|C],C1).
elimina([T|C],[T|C1]):-
     elimina(C,C1).
```

**Esercizio 83.**

Si consideri una qualsiasi formula proposizionale contenente i soli connettivi logici di negazione, congiunzione e disgiunzione. Avendo predefinito i seguenti operatori:

```
:-  op(30,fx,non).
:-  op(100,xfy,or).
:-  op(100,xfy,and).
```

rappresentiamo detta formula proposizionale mediante una struttura Prolog in cui le lettere proposizionali sono rappresentate da atomi Prolog (es.: `(a or non (b and c))`)

Scrivere un programma Prolog `contraddizione(FP)` che prenda in `FP` una struttura Prolog rappresentante una formula proposizionale ed esca con successo se la `FP` è una contraddizione, altrimenti fallisca.

```
?- contraddizione(non((a or non a)and(b or non b))).
true.
?- contraddizione(a and (c or b or non a)).
false.
```

SOLUZIONE

```
valore(f,f):-!.
valore(v,v):-!.

valore(non v, f):- !.
valore(non f, v):- !.

valore(v and v, v):- !.
valore(v and f, f):- !.
valore(f and v, f):- !.
valore(f and f, f):- !.

valore(v or v, v):- !.
valore(v or f, v):- !.
valore(f or v, v):- !.
valore(f or f, f):- !.

valore(non A, X):-
      valore(A,A1),
      valore(non A1, X).
valore(A and B, X):-
      valore(A,A1),
      valore(B,B1),
      valore(A1 and B1, X).
valore(A or B, X):-
      valore(A,A1),
      valore(B,B1),
      valore(A1 or B1, X).

contraddizione(FP):-
      proposizioni(FP,L),
      findall(V,(genera(L,L1),sostituisci(FP,L,L1,V)),ListaVal),
      nl,write(ListaVal),write('   '),
      tutto_falso(ListaVal).

proposizioni(F,L):-
      setof(X,foglia(F,X),L).

foglia(F,F):-
      atomic(F).
foglia(F,L):-
      \+ atomic(F),
      F =.. [_|Args],
      member(X,Args),
      foglia(X,L).

genera([_|C],[X|C1]):-
      member(X,[v,f]),
```

```
      genera(C,C1).
genera([],[]).

sostituisci(F,[T1|C1],[T2|C2],V):-
      substitute(T1,T2,F,NF),
      sostituisci(NF,C1,C2,V).
sostituisci(NF,[],[],V):-
      valore(NF,V).

substitute(Old,New,Old,New).
substitute(Old,_,Term,Term):-
       atomic(Term),
       Term \= Old.
substitute(Old,New,Term,Term1):-
       not(atomic(Term)),
       functor(Term,F,N),
       functor(Term1,F,N),
       substitute(N,Old,New,Term,Term1).

substitute(N,Old,New,Term,Term1):-
       N > 0,
       arg(N,Term,Arg),
       substitute(Old,New,Arg,Arg1),
       arg(N,Term1,Arg1),
       N1 is N - 1,
       substitute(N1,Old,New,Term,Term1).
substitute(0,_,_,_,_).

tutto_falso([]).
tutto_falso([f|C]):-
       tutto_falso(C).
```

SOLUZIONE2

```
% si privilegia l'efficienza alla completezza dell'elenco dei modelli
% (che non è richiesto), risolvendo gli operatori and e or in maniera
% più rapida possibile (valutando un solo elemento alla volta, quando
% possibile). Evitato l'uso del cut dove non necessario.

contraddizione(Formula):-
      trovaatomi(Formula,Atomiconrip),
      setof(X,member(X,Atomiconrip),Atomi), %elimina le ripetizioni
      findall(Valutaz,
            (   valutaatomi(Atomi,Valutaz),valutaformula(Formula,Valutaz,true)
            ),
            Ris),
      (   Ris=[],write('E\' una contraddizione perché non esiste alcun modello.'),true,!;

      Ris=[T|_],write('Non è una contraddizione perché la formula ha almeno un
modello.'),nl,write('Il primo modello trovato è: '),write(T),fail
      ).

trovaatomi(non X,A):-
      trovaatomi(X,A).
trovaatomi(X or Y,A):-
      trovaatomi(X,A1),trovaatomi(Y,A2),
      append(A1,A2,A).
trovaatomi(X and Y,A):-
      trovaatomi(X,A1),trovaatomi(Y,A2),
      append(A1,A2,A).
trovaatomi(A,[A]):-
      atom(A).

valutaatomo(true).
valutaatomo(false).

valutaatomi([],[]).
valutaatomi([T|C],[T/V|C2]):-
      valutaatomo(V),
      valutaatomi(C,C2).
```

```
valutaformula(non A,V,true):-
      valutaformula(A,V,false).
valutaformula(non A,V,false):-
      valutaformula(A,V,true).

valutaformula(A and B,V,false):-
      valutaformula(A,V,false);
      valutaformula(B,V,false).
valutaformula(A and B,V,true):-
      valutaformula(A,V,true),
      valutaformula(B,V,true).

valutaformula(A or B,V,true):-
      valutaformula(A,V,true);
      valutaformula(B,V,true).
valutaformula(A or B,V,false):-
      valutaformula(A,V,false),
      valutaformula(B,V,false).


valutaformula(X,V,Valoreatomo):-
      atom(X),
      member(X/Valoreatomo,V).
```

**Esercizio 84.**

Scrivere un predicato Prolog `max_seqs(L)`, dove `L` è una lista di interi, che stampi la più estesa sequenza crescente e la più estesa sequenza decrescente.

```
?- max_seqs([1,2,1,2,3,4,2,1,2,1]).
max sequenza crescente [1, 2, 3, 4]
max sequenza decrescente [4, 2, 1]
true

?- max_seqs([1,2,1,3,1,3,4,5,6,3,2,1,2,3,2,1]).
max sequenza crescente [1, 3, 4, 5, 6]
max sequenza decrescente [6, 3, 2, 1]
true
```

SOLUZIONE:

```
crescente0([T|C]):-
   cre(T,C).

cre(_,[]):-
    !.
cre(X,[T|_]):-
    X>T,
    !,
    fail.
cre(X,[T|C]):-
    X < T,
    !,
    cre(T,C).

decrescente0([T|C]):-
   decre(T,C).

decre(_,[]):-
    !.
decre(X,[T|_]):-
    X<T,
    !,
    fail.
decre(X,[T|C]):-
    X > T,
    !,
    decre(T,C).

sottolista(S,L):-
     append(P,_,L),
     append(_,S,P).

crescente(L1,S):-
     sottolista(S,L1),
     crescente0(S).

decrescente(L1,S):-
     sottolista(S,L1),
     decrescente0(S).   % e non \+ crescente0(S).

max_crescente(L1,L2):-
     findall(X, crescente(L1,X),L),
     mass(L,L2).

max_decrescente(L1,L2):-
     findall(X, decrescente(L1,X),L),
     mass(L,L2).

max_seqs(L):-
     max_crescente(L,LCRESC),
```

```
     write('max sequenza crescente '),
     write(LCRESC),nl,
     max_decrescente(L,LDECRESC),
     write('max sequenza decrescente '),
     write(LDECRESC),nl.

mass(L,L2):-
     mass0(L,_,0,L2).

mass0([],L2,_,L2).
mass0([T|C],LPROVV,N,L2):-
     length(T,M),
     M=<N,
     mass0(C,LPROVV,N,L2).
mass0([T|C],_,N,L2):-
     length(T,M),
     M>N,
     mass0(C,T,M,L2).
```

**Esercizio 85.**

Scrivere un predicato Prolog `cartprod(X)`, che legge dall'input due qualunque liste e, dopo aver controllato che effettivamente siano liste ben formate (di qualunque cosa), restituisca in `X` il loro prodotto cartesiano.

SOLUZIONE

```
cartprod(Z):-
     read(X),
     islist(X),
     read(Y),
     islist(Y),
     findall((A,B),(member(A,X),member(B,Y)),Z).

islist([]).
islist([_|C]):-
     islist(C).
```

**Esercizio 86.**

Definire un predicato Prolog `take(List,N,Result)` che ha successo se e solo se `Result` è la lista costituita dai primi `N` elementi di `List` (nello stesso ordine). Se `List` ha meno di `N` elementi o se `N` è zero o negativo, il goal `take(List,N,Result)` fallisce segnalando il tipo di errore.
Definire inoltre un predicato `drop(List,N,Result)` che ha successo se e solo se `Result` è la lista costituita dagli elementi di `List` (nello stesso ordine) che rimangono quando sono stati eliminati i primi `N` elementi. Se `List` ha meno di `N` elementi o se `N` è zero o negativo, il goal `drop(List,N,Result)` fallisce segnalando il tipo di errore.

```
?- take([a,b,c,d,e,f],4,Result).
 Result = [a, b, c, d].
?- take([a,b,c,d,e,f],-4,Result).
ERRORE:  N non positivo
false.
?- take([a,b,c,d,e,f],8,Result).
ERRORE:  La lista ha meno di 8 elementi!!
false.

?- drop([a,b,c,d,e,f],4,Result).
Result = [e, f].
?- drop([a,b,c,d,e,f],-4,Result).
ERRORE:  N non positivo
false.
?- drop([a,b,c,d,e,f],8,Result).
ERRORE:  La lista ha meno di 8 elementi!!
false.

SOLUZIONE
take(List,N,_):-
    length(List,K),
    K<N,
    write('ERRORE:  La lista ha meno di '),
    write(N),
    write(' elementi!!'),
    !,
    fail.
take(_,N,_):-
    N=<0,
    write('ERRORE:  N non positivo'),
    !,
    fail.
take(List,N,Result):-
    take1(List,N,Result).

take1(_,0,[]):-
    !.
take1([T|C],N,[T|C1]):-
    N1 is N-1,
    take1(C,N1,C1).

drop(List,N,_):-
    length(List,K),
    K<N,
    write('ERRORE:  La lista ha meno di '),
    write(N),
    write(' elementi!!'),
    !,
    fail.
drop(_,N,_):-
    N=<0,
    write('ERRORE:  N non positivo'),
```

```
    !,
    fail.
drop(List,N,Result):-
    drop1(List,0,N,Result).

drop1(L,N,N,L):-
    !.
drop1([_|C],NCONT,N,C1):-
    NCONT1 is NCONT+1,
    drop1(C,NCONT1,N,C1).
```

**Esercizio 87.**

Una giuria composta da due commissioni deve assegnare un premio letterario. I voti della prima commissione sono messi nel file VOTI1.DAT e quelli della seconda in VOTI2.DAT. I file contengono uno per riga il cognome e il nome del candidato votato da ciascun commissario e non è nota a priori la lunghezza del file stesso.
Scrivere un programma che legga i voti da entrambi i file, costruisca una lista ordinata in ordine di voti ricevuti, con il relativo totale dei voti, ed effettui la stampa della lista così ottenuta.
Il programma deve prevedere la correzioni di errori di assegnazioni di voti, per cui chiede il cognome di chi ha avuto erroneamente i voti e il cognome di chi non ne avuti e l'ammontare dei voti (vedi esempio). Avvenuta la correzione venga stampata la graduatoria corretta.
I cognomi sono tutti diversi. Esempio:

| *File: VOTI1.DAT* | *File: VOTI2.DAT* |
|---|---|
| `a(rossi,franco).`<br>`a(rossi,franco).`<br>`a(armillei,franco).`<br>`a(farneti,luca).`<br>`a(bianchi,andrea).`<br>`a(rossi,franco).`<br>`a(armillei,franco).`<br>`a(farneti,luca)` | `b(rossi,franco).`<br>`b(rossi,franco).`<br>`b(rossi,franco).`<br>`b(rossi,franco).`<br>`b(farneti,luca).`<br>`b(rossi,franco).`<br>`b(armillei,franco).`<br>`b(farneti,luca).`<br>`b(neri,luigi).` |

```
?- go.
rossi  franco    8
farneti  luca    4
armillei  franco    3
bianchi  andrea    1
neri  luigi    1

Correzione (basta il cognome)
da chi devo togliere i voti? rossi.
a chi devo darli? armillei.
Quanti voti? 3.

armillei  franco    6
rossi  franco    5
farneti  luca    4
bianchi  andrea    1
neri  luigi    1
true.
```

**SOLUZIONE**

```
go:-
  lista1(L1),
  lista2(L2),
  append(L1,L2,L),
  ris(L,RIS),
  quick_decresc(RIS,RIS_ORD),
  stampa(RIS_ORD),
  correzione(RIS_ORD,LA),
  quick_decresc(LA,LORD),
  stampa(LORD).

lista1(Lines):-
    open('C:/LETTURA DA FILE/PremioLett/voti1.dat', read, Str),
    read_file1(Str,Lines),
```

```
    close(Str).

read_file1(Stream,[]) :-
    at_end_of_stream(Stream),
    !.
read_file1(Stream,[X|L]) :-
    \+ at_end_of_stream(Stream),
    read(Stream,X1),
    X1=a(XX,Y),
    X=x(XX,Y),
    read_file1(Stream,L).

lista2(Lines):-
    open('C:/LETTURA DA FILE/PremioLett/voti2.dat', read, Str),
    read_file2(Str,Lines),
    close(Str).

read_file2(Stream,[]) :-
    at_end_of_stream(Stream),
    !.
read_file2(Stream,[X|L]) :-
    \+ at_end_of_stream(Stream),
    read(Stream,X1),
    X1=b(XX,Y),
    X=x(XX,Y),
    read_file2(Stream,L).


ris([],[]).
ris([T|C],[T1|Resto]):-
   T=x(Cogn,Nome),
   T1=x1(Cogn,Nome,N),
   ris0(T,C,C1,N),
   ris(C1,Resto).

ris0(T,L,L1,N):-
   cancella_t(L,T,L1,1,N).

 cancella_t([],_,[],N,N).
 cancella_t([X|C],X,C1,N,N2):-
     !,
     N1 is N+1,
     cancella_t(C,X,C1,N1,N2).
 cancella_t([X|C],Y,[X|C1],N,N2):-
     cancella_t(C,Y,C1,N,N2).


quick_decresc([],[]).
quick_decresc([N|C],L):-
      split(N,C,Lmin,Lmagg),
      quick_decresc(Lmin,Lmin_ord),
      quick_decresc(Lmagg,Lmagg_ord),
      append(Lmagg_ord,[N|Lmin_ord],L).

split(_,[],[],[]).
split(N,[N1|C],[N1|L1],L2):-
      N = x1(_,_,Voti),
      N1 = x1(_,_,Voti1),
      Voti1 =< Voti,
      !,
      split(N,C,L1,L2).
split(N,[N1|C],L1,[N1|L2]):-
      split(N,C,L1,L2).


stampa([]).
```

```
stampa([T|C]):-
    T = x1(Cogn,Nome,Voti),
    write(Cogn),tab(2),write(Nome),tab(4),write(Voti),nl,
    stampa(C).

correzione(L,L1):-
  nl,write('Correzione (basta il cognome)'),nl,
  write('da chi devo togliere i voti? '),read(Cogn1),
  write('a chi devo darli? '),read(Cogn2),
  write('Quanti voti? '), read(Voti),
  opera(Cogn1,Cogn2,Voti,L,L1).

opera(Cogn1,Cogn2,Voti,L,L1):-
     cancella(Cogn1,Voti,L,L2),
     aggiungi(Cogn2,Voti,L2,L1).

cancella(Cogn1,Voti,[T|C],[T1|C]):-
     T=x1(Cogn1,NOME,N),
     !,
     N1 is N - Voti,
     T1=x1(Cogn1,NOME,N1).
cancella(Cogn1,Voti,[T|C],[T|C1]):-
     cancella(Cogn1,Voti,C,C1).

aggiungi(Cogn1,Voti,[T|C],[T1|C]):-
     T=x1(Cogn1,NOME,N),
     !,
     N1 is N + Voti,
     T1=x1(Cogn1,NOME,N1).
aggiungi(Cogn1,Voti,[T|C],[T|C1]):-
     aggiungi(Cogn1,Voti,C,C1).
```

**Esercizio 88.**

Scrivere un predicato Prolog `somma(LL, N)`, dove `LL` è istanziata ad una lista di liste, indentata a più livelli, di interi che istanzi `N` alla somma si tutti gli interi di contenuti in `LL`.

```
?- somma([[2,3,[1],[[],4]],5],N).
N = 15.

?- somma([[2,3,[1],[[6]]],5],N).
N = 17.
```

**SOLUZIONE**

```
somma([],0).
somma([X|Y],N):-
    lista(X),
    !,
    somma(X,N1),
    somma(Y,N2),
    N is N1 + N2.
somma([X|Y],N):-
    somma(Y,N1),
    N is N1 + X.

lista([]).
lista([_|C]):-
    lista(C).
```

## Esercizio 89.

Il file di testo *volo.txt* contiene fatti del tipo `volo/3` relativi alle prenotazioni di un volo di linea, per esempio:

```
volo(andrea,rossi,1).
volo(anna,pinosi,1).
volo(carlo,mari,0).
volo(sandro,lunati,1).
volo(paolo,amaldi,0).
volo(rosanna,vieri,0).
volo(eugenia,buffon,0).
volo(maria,castelli,1).
```

dove il terzo atomo rappresenta la posizione in lista come "stand-by" (`0`) o confermato (`1`). L'ordine con cui i clienti in stand-by compaiono nel database riflette implicitamente la loro priorità acquisita. Si scriva un programma PROLOG che legga il file e inserisca i dati letti rispettivamente in due liste:

4) L1 che contenga l'elenco dei nominativi, con funtore a due argomenti (nome e cognome), dei clienti ***confermati***, ordinata rispetto al cognome
5) L2 che contenga l'elenco dei clienti in stand-by, con funtore a tre argomenti (nome, cognome e priorità).

Suggerimento: si definisca un predicato, `setta_priorita(NONConfermati,NONConfermati1,N)` che assegni la priorità ai clienti in stand-by, utilizzando un intero in modo che un numero minore corrisponda a una maggiore priorità).

Il programma stampi poi le due liste, dopodiché:

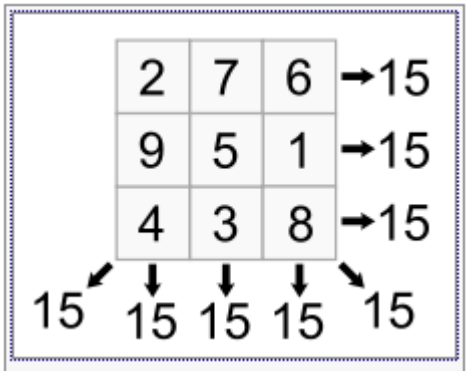
1. legga dallo standard input il nome ed il cognome di un cliente nella lista L1
2. lo elimini da detta lista
3. estragga da L2 il cliente con maggior priorità
4. inserisca quest'ultimo in L1
5. stampi la lista dei nuovi confermati in ordine alfabetico.

**Esempio**: nel caso del file *volo.txt* in esempio l'esecuzione potrebbe essere quella che segue:

```
?- go.
LISTA DEI CONFERMATI
castelli  maria
lunati  sandro
pinosi  anna
rossi  andrea

LISTA DEI -NON- CONFERMATI
mari  carlo   1
amaldi  paolo   2
vieri  rosanna   3
buffon  eugenia   4

--> Dai il nominativo della persona che rinuncia
NOME anna.
COGNOME pinosi.
LISTA DEI CONFERMATI AGGIORNATA
castelli  maria
lunati  sandro
mari  carlo
rossi  andrea
true
```

**Esercizio 90.**

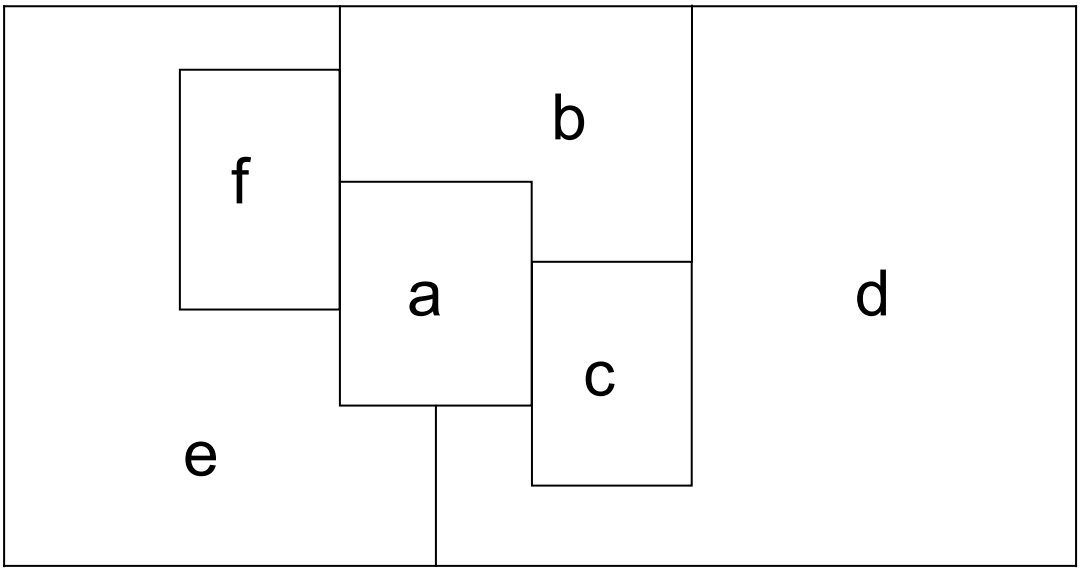
Un **quadrato magico** è uno schieramento di numeri interi distinti in una tabella quadrata tale che la somma dei numeri presenti in ogni riga, in ogni colonna e in entrambe le diagonali dia sempre lo stesso numero. Scrivere un programma Prolog **`quadrati_magici(L)`** che trovi, in risoddisfacimento, i quadrati magici 3x3 con gli interi da 1 a 9, di cui uno è riportato in figura.




```
?- quadrati_magici(L).
L = [[2, 7, 6], [9, 5, 1], [4, 3, 8]] ;
L = [[2, 9, 4], [7, 5, 3], [6, 1, 8]] ;
L = [[4, 3, 8], [9, 5, 1], [2, 7, 6]] ;
L = [[4, 9, 2], [3, 5, 7], [8, 1, 6]] ;
L = [[6, 1, 8], [7, 5, 3], [2, 9, 4]] ;
L = [[6, 7, 2], [1, 5, 9], [8, 3, 4]] ;
L = [[8, 1, 6], [3, 5, 7], [4, 9, 2]] ;
L = [[8, 3, 4], [1, 5, 9], [6, 7, 2]] ;
false.
```

**Esercizio 91.**

Si risolva la colorazione della seguente mappa senza far ricorso alla **Constraint Logic Programming. [14 punti]**




Si deve avere un output del tipo

```
?- test_color(test,Map).
La regione a viene colorata di red
La regione b viene colorata di yellow
La regione c viene colorata di blue
La regione d viene colorata di white
La regione e viene colorata di blue
La regione f viene colorata di white


Map = [region(a, red, [yellow, blue, white, blue, white]),……..
```

**Esercizio 92.**
Scrivere un programma Prolog **somme/2** che avendo il primo argomento istanziato ad una lista di interi, istanzi la seconda lista alle somme rispettivamente dell'ultimo elemento, degli ultimi due elementi, degli ultimi tre elementi ecc. come mostrato dall'esempio. **[6 punti]**

```
?- somme([1,2,3,4,5],L).

L = [5, 9, 12, 14, 15] ;

No
```

**Esercizio 93.**

Il problena del knapsack consiste nel massimizzare il valore degli oggetti che si inseriscono in uno zaino con il vincolo che il volume totale degli oggetti non superi la capacità dello zaino.
Si consideri la seguente istanza del problema del knapsack. Lo zaino ha dimensione 38.
Gli oggetti sono: il whisky che occupa 5, vale 15 e ce ne sono 2, il profumo che occupa 4,vale 10 e ce sono 6, il cioccolato che occupa 3 vale 6 e ce ne sono 10 e le sigarette che occupano 2 e valgono 7 e ce sono 8. C'è l'obbligo di almeno un whisky, e 2 cioccolate, mentre non c'è nessun obbligo per gli altri oggetti. Si ricerca un'allocazione dello zaino che assicuri un profitto di almeno 100. Qual è il contenuto del sacco che massimizza e quanto vale?
Risolvere il problema facendo uso della **Constraint Logic Programming** e quindi del modulo **clpfd. [10 punti]**

```
?- saccomax.
il sacco max si ha con 2 wisky  2 profumo  2 cioccolato  7 sigarette
per un valore di 111
true.
```

## Esercizio 94.

Scrivere un predicato **go/0** che ***utilizzando la libreria del CLP*** stampi una soluzione, se esiste, per il problema delle N regine, con N da 1 a 15, come sotto riportato.

```
?- go.
con 1 regine[1]
con 2 regine  no soluzioni
con 3 regine  no soluzioni
con 4 regine[2,4,1,3]
con 5 regine[1,3,5,2,4]
con 6 regine[2,4,6,1,3,5]
con 7 regine[1,3,5,7,2,4,6]
con 8 regine[1,5,8,6,3,7,2,4]
con 9 regine[1,3,6,8,2,4,9,7,5]
con 10 regine[1,3,6,8,10,5,9,2,4,7]
con 11 regine[1,3,5,7,9,11,2,4,6,8,10]
con 12 regine[1,3,5,8,10,12,6,11,2,7,9,4]
con 13 regine[1,3,5,2,9,12,10,13,4,6,8,11,7]
con 14 regine[1,3,5,7,12,10,13,4,14,9,2,6,8,11]
true
```

**SOLUZIONE**

```
:- use_module(library(clpfd)).

go:-b(1,16).

b(N,N):-!.
b(N1,N):-
    N1<N,
    write('con '),write(N1),write(' regine'),
    queens(N1, Queens),
    stampa(Queens),nl,
    N2 is N1+1,
    b(N2,N).

stampa(X):-
    var(X),!,
    write(' ').
stampa(X):-
    write(X).

queens(N, Queens) :-
     length(Queens, N),
     Queens ins 1..N,
     constrain(Queens),
     labeling([], Queens).
queens(N, Queens):-
     write('  no soluzioni').
constrain(Queens) :-
     all_different(Queens),
     diagonal(Queens).

diagonal([]).
diagonal([Q|Queens]) :-
     sicure(Q, 1, Queens),
     diagonal(Queens).

sicure(_,_,[]).
sicure(X,D,[Q|Queens]) :-
     nonattacca(X,Q,D),
     D1 is D+1,
     sicure(X,D1,Queens).

nonattacca(X,Y,D) :-
     X + D #\= Y,
     Y + D #\= X.
```

## Esercizio 95.

Il file `memo.dat` contiene i dati relativi agli appuntamenti giornalieri di uno studio legale. Ciascun appuntamento è caratterizzato dal cognome della persona da incontrare e l'ora (dalle 7 alle 13). Es:

```
memo(verdi,11).
memo(rossi,7).
memo(bianchi,9).
memo(blu,7).
memo(neri,11).
memo(carli,11).
memo(rapini,8).
```

Si scriva un programma PROLOG che:

- realizzi una lista di liste; la dimensione della lista è pari al numero delle ore di ricevimento della giornata e ciascun elemento contiene l'ora e i nominativi (ordinati alfabeticamente) delle persone che verranno ricevute
- letto un cognome dallo standard input, visualizzi a quale ora tale persona verrà ricevuta o un opportuno messaggio se per quella persona non è previsto alcun appuntamento
- stampi sul file `agenda.dat` gli elenchi dei nominativi, ciascuno preceduto dall'ora di ricevimento

ESEMPIO

```
?- go.
dai il cognome
|: verdi.
verdi ha appuntamento per le 11
true.
?- go.
dai il cognome
|: nessuno.
Nessun appuntamento per nessuno
true
```

`agenda.dat`

```
*** ORE 7 ***
blu
rossi
*** ORE 8 ***
rapini
*** ORE 9 ***
bianchi
*** ORE 11 ***
carli
neri
verdi
```

**SOLUZIONE**

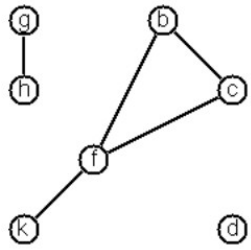
```
% solo ore 7-13

go:-
    prepara(7,Lore),
    aggiorna(Lore,LoreN),
    ricevimento(LoreN),
    stampa(LoreN).

prepara(14,[]):- % il pomeriggio lo studio legale non riceve
    !.
prepara(ORA,[ric(ORA,[])|C]):- % crea un'agenda vuota
    ORA1 is ORA +1,
    prepara(ORA1,C).

aggiorna(Lore,LoreN):- % aggiorna l''agenda con i dati letti dal file memo.dat
    open('memo.dat', read, Str),
```

```
    read_file(Str,Lines),
    close(Str),
    aggiorna0(Lore,Lines,LoreN).

read_file(Stream,[]) :-
    at_end_of_stream(Stream),
    !.

read_file(Stream,[X|L]) :-
    \+ at_end_of_stream(Stream),
    read(Stream,X),
    read_file(Stream,L).

aggiorna0([],_,[]):-
    !.
aggiorna0([T|C],Lines,[T1|C1]):-
    T = ric(ORA,L),
    tuttiOra(ORA,Lines,[],LClienti),
    append(L,LClienti,LL),
    T1 = ric(ORA,LL),
    aggiorna0(C,Lines,C1).

tuttiOra(_,[],LF,LF):-
    !.
tuttiOra(ORA,[T|C],LP,LF):-
    T = memo(NOME,ORA),
    !,
    LP1 = [NOME|LP],
    tuttiOra(ORA,C,LP1,LF).
tuttiOra(ORA,[_|C],LP,LF):-
    tuttiOra(ORA,C,LP,LF).

ricevimento(LoreN):-
    write('dai il cognome'),nl,
    read(NOME),
    cerca(NOME,LoreN,N),
    stampa(NOME,N).

cerca(NOME,LoreN,N):-
    cerca0(NOME,LoreN,N).

cerca0(_,[],non_trovato):-
    !.
cerca0(NOME,[T|_],N):-
    T = ric(M,L),
    member(NOME,L),
    !,
    N = M.
cerca0(NOME,[_|C],N):-
    cerca0(NOME,C,N).

stampa(NOME,non_trovato):-
    write('Nessun appuntamento per '),
    write(NOME),nl.
stampa(NOME,N):-
    write(NOME),
    write(' ha appuntamento per le '),
    write(N),
    nl.

stampa(L):-
    open('agenda.dat', write, Str),
    stampa0(Str,L),
    close(Str).

stampa0(_,[]):-
    !.
stampa0(Str,[T|C]):-
    T =ric(_,L),
    L = [],
```

```
    !,
    stampa0(Str,C).
stampa0(Str,[T|C]):-
    T =ric(N,L),
    write(Str,'*** ORE '),write(Str,N),write(Str,' ***'),nl(Str),
    stampaL(Str,L),
    stampa0(Str,C).

stampaL(_,[]):-
    !.
stampaL(Str,[T|C]):-
    write(Str,T),nl(Str),
    stampaL(Str,C).
```

## Esercizio 96.

Si supponga di rappresentare i grafi




in questo modo:

`grafo([b,c,d,f,g,h,k],[e(b,c),e(b,f),e(c,f),e(f,k),e(g,h)]).`

Due grafi G1(N1,E1) e G2(N2,E2) si dicono ***isomorfi*** se esiste una corrispondenza biettiva

f: N1 → N2

tale per cui per ogni coppia di nodi X,Y in N1, X e Y sono adiacenti se e solo se anche f(X) e f(Y) sono adiacenti.

Scrivere un predicato:

`iso(grafo(Nodi1,Archi1),grafo(Nodi2,Archi2))`

che restituisca `true` se e solo se i due grafi (istanziati) sono *isomorfi*.

ESEMPIO

```
?- iso(grafo([b,c,d,f,g,h,k],
             [e(b,c),e(b,f),e(c,f),e(f,k),e(g,h)]),
       grafo([b,c,d,f,g,h,k],
             [e(c,f),e(f,k),e(g,h),e(b,f),e(c,b)])).
true

?- iso(grafo([b,c,d,f,g,h,k],
             [e(b,c),e(b,f),e(c,f),e(f,k),e(g,h)]),
       grafo([2,3,4,5,6,7,8],
             [e(2,3),e(3,5),e(5,8),e(6,7),e(2,5)])).
true

?- iso(grafo([b,c,d,f,g,h,k],
             [e(b,c),e(b,f),e(c,f),e(f,k),e(g,h)]),
       grafo([2,3,4,5,6,7,8],
             [e(2,3),e(4,5),e(5,8),e(6,7),e(2,5)])).
false
```

**SOLUZIONE**

```
iso(grafo(Nodi1,Archi1),grafo(Nodi2,Archi2)) :- % due grafi sono isomorfi
       corrispondenza_biettiva(Nodi1,Nodi2,Corr), % se esiste una corrispondenza biettiva fra i loro nodi
       isomorfi(Archi1,Archi2,Corr). % alla luce della quale i due grafi sono isomorfi

corrispondenza_biettiva([],[],[]). % fra due insiemi vuoti c'e' una corrispondenza vuota
corrispondenza_biettiva([T|C],Nodi,[c(T,N)|CC]) :- % il primo elemento N della corrispondenza
       select(N,Nodi,Resto), % viene selezionato non deterministicamente, producenzo un Resto
       corrispondenza_biettiva(C,Resto,CC). % dal quale saranno selezionati i successivi elementi

isomorfi([],[],_). % due grafi vuoti sono isomorfi a prescindere dalla corrispondenza
isomorfi([e(X,Y)|Sottografo],Archi,Corr) :- % due grafi sono isomorfi se tolto un arco e(X,Y) dal primo
       select(e(X1,Y1),Archi,Sottografo1), % selezionato un arco e(X1,Y1) del secondo, producendo un Sottografo1
       member(c(X,X1),Corr), % si dimostra che X e' in corrispondenza con X1
       member(c(Y,Y1),Corr), % si dimostra che Y e' in corrispondenza con Y1
       isomorfi(Sottografo,Sottografo1,Corr). % e si dimostra che i due sottografi sono isomorfi
```

## Esercizio 97.

Scrivere un predicato PROLOG `cambia(L1, L2, L3, L4)`, dove:

- `L1` è una lista di atomi PROLOG consistenti in una lettera
- `L2` è una lista di elementi del tipo `a/b` in cui `a` è dello stesso tipo di atomi di `L1` e `b` un intero
- `L3` è la lista che si ottiene cambiando in `L1` ogni occorrenza dell'elemento `a` di `L1` con `b`. Qualora `a` non fosse presente in `L2` lo si sostituisca con `0`. Gli elementi di `L1` possono essere ripetuti. In `L2` possono essere presenti degli `a/b` con `a` non presente in `L1`; tali `a/b` vanno restituiti nella lista `L4`.

Esempio.

```
?- cambia([a,d,s,a,c,t],[d/8,t/3,q/7,s/2,u/9],L3,L4).
L3 = [0, 8, 2, 0, 0, 3],
L4 = [q/7, u/9].
```

SOLUZIONE

```
% cambia([a,d,s,a,c,t],[d/8,t/3,q/7,s/2,u/9],L3,L4).


cambia([],S,[],S):-
    !.
cambia([T|C],S,[T1|C1],R):-
    member(T/T1,S),
    !,
    delete(S,T/T1,S1),
    cambia(C,S1,C1,R).
cambia([_|C],S,[0|C1],R):-
    !,
    cambia(C,S,C1,R).
```

## Esercizio 98.

1. Ci sono cinque case.
2. L'inglese vive nella casa rossa.
3. Lo spagnolo possiede il cane.
4. Nella casa verde si beve caffè.
5. L'ucraino beve tè.
6. La casa verde è immediatamente a destra della casa avorio.
7. Il fumatore di Winston possiede lumache.
8. Nella casa gialla si fumano le Kools.
9. Nella casa di mezzo si beve il latte.
10. Il norvegese vive nella prima casa.
11. L'uomo che fuma Chesterfield abita nella casa accanto a quella con la volpe.
12. Nella casa accanto alla casa dove è conservato il cavallo si fumano le Kools.
13. Il fumatore di Lucky Strike beve succo d'arancia.
14. Il giapponese fuma le Parliaments.
15. Il norvegese vive vicino alla casa blu.

Chi beve l'acqua? Chi possiede la zebra?

Scrivere un programma PROLOG `risolvi(PossessoreZebra,BevitoreAcqua)` che, tenendo in debito conto le indicazioni fornite, istanzi le due variabili in maniera da rispondere alle due domande del caso.

**SOLUZIONE**

```
risolvi(PossessoreZebra, BevitoreAcqua):-
  indizi(Caseggiato),
  domande(Caseggiato, PossessoreZebra, BevitoreAcqua).

indizi(Caseggiato):-
  abitazione(A, Caseggiato), colore(A, rosso), nationalita(A, inglese),         /*  1 */
  abitazione(B, Caseggiato), nationalita(B, spagnolo), animale(B, cane),        /*  2 */
  abitazione(C, Caseggiato), colore(C, verde), beve(C, caffe),                  /*  3 */
  abitazione(D, Caseggiato), nationalita(D, ucraino), beve(D, tea),             /*  4 */
  in_mezzo_a_destra(Caseggiato, E, F), colore(E, verde), colore(F, avorio),     /*  5 */
  abitazione(G, Caseggiato), fuma(G, winston), animale(G, lumache),             /*  6 */
  abitazione(H, Caseggiato), fuma(H, kools), colore(H, giallo),                 /*  7 */
  in_mezzo(Caseggiato, I), beve(I, latte),                                      /*  8 */
  prima(Caseggiato, J), nationalita(J, norvegese),                              /*  9 */
  vicino_a(Caseggiato, K, L), fuma(K, chesterfields), animale(L, volpe),        /* 10 */
  vicino_a(Caseggiato, M, N), fuma(M, kools), animale(N, cavallo),              /* 11 */
  abitazione(O, Caseggiato), fuma(O, luckystrike), beve(O, succodarancia),      /* 12 */
  abitazione(P, Caseggiato), nationalita(P, giapponese), fuma(P, parliaments),  /* 13 */
  vicino_a(Caseggiato, Q, R), nationalita(Q, norvegese), colore(R, blu).        /* 14 */

domande(Caseggiato, PossessoreZebra, BevitoreAcqua):-
  abitazione(X, Caseggiato), animale(X, zebra), nationalita(X, PossessoreZebra), %  giapponese
  abitazione(Y, Caseggiato), beve(Y, water), nationalita(Y, BevitoreAcqua). %  norvegese

colore(casa(C,_,_,_,_), C).
nationalita(casa(_,N,_,_,_), N).
animale(casa(_,_,P,_,_), P).
beve(casa(_,_,_,D,_), D).
fuma(casa(_,_,_,_,S), S).

prima(caseggiato(X,_,_,_,_), X).

in_mezzo(caseggiato(_,_,X,_,_), X).

in_mezzo_a_destra(caseggiato(L,R,_,_,_), R, L).
in_mezzo_a_destra(caseggiato(_,L,R,_,_), R, L).
in_mezzo_a_destra(caseggiato(_,_,L,R,_), R, L).
in_mezzo_a_destra(caseggiato(_,_,_,L,R), R, L).

vicino_a(Xs, X, Y):- in_mezzo_a_destra(Xs, X, Y).
vicino_a(Xs, X, Y):- in_mezzo_a_destra(Xs, Y, X).
```

```
abitazione(casa(A,B,C,D,E), caseggiato(casa(A,B,C,D,E),_,_,_,_)).
abitazione(casa(A,B,C,D,E), caseggiato(_,casa(A,B,C,D,E),_,_,_)).
abitazione(casa(A,B,C,D,E), caseggiato(_,_,casa(A,B,C,D,E),_,_)).
abitazione(casa(A,B,C,D,E), caseggiato(_,_,_,casa(A,B,C,D,E),_)).
abitazione(casa(A,B,C,D,E), caseggiato(_,_,_,_,casa(A,B,C,D,E))).
```