Esercizio 1. Svolgere tutti i punti.

a-1) Si consideri il seguente programma logico e se ne calcolino gli answer set, illustrando adeguatamente il procedimento seguito.

```
a(Y) v b(Z) :- not c(Y), d(Y,Z).  2
c(Y) | b(Y) :- d(Y,_), not a(Y).  1
d(X,Z) :- c(Y),  a(X), Z=X+Y.    3
d(3,1).
d(2,4).
a(2).
```

a-2) Si aggiunga il seguente strong constraint al programma del punto precedente.

```
:-#sum{X,Y:d(X,Y)}<5.
```

a-1)

~~c(2) | b(2) :- d(2,4), not a(2).~~
c(3) | b(3) :- ~~d(3,1)~~, not a(3).

~~a(2) | b(4) :- not c(2), d(2,4).~~
a(3) | b(1) :- not c(3), ~~d(3,1)~~.

* b(4) non sarà mai supportato perché a(2) EDB

d(2,5) :- c(3), ~~a(2)~~, 5=2+3.
~~d(3,6) :- c(3), a(3), 6=3+3.~~

* c(3) e a(3) non saranno mai veri contemporaneamente

* STEP 2
* La seconda regola non genererebbe più nulla

~~a(2) | b(5) :- not c(2), d(2,5).~~
~~a(3) | b(6) :- not c(3), d(3,6).~~

* IDEM COME SOPRA
* d(3,6) non sarà mai vero

AS: {
A1: { [... EDB ...], b(3), b(1) }
A2: { [... EDB ...], c(3), d(2,5) }
A3: { [... EDB ...], a(3) }

}

a-2)

:- #sum{X,Y: d(X,Y)} < 5.

Inserendo questo strong constraint, il programma rimane invariato poiché la sola SUM dei fatti d(X,Y) degli EDB è uguale a 5, quindi rendendo "innocuo" il constraint.

b) Si consideri ora un programma P (non è necessario sapere come è fatto) i cui answer set sono già stati calcolati e sono riportati di seguito.

```
A1: {a(2,1,3), c(1), c(2), m(1,1), t(2,1), m(2,1), m(2,2), f(1), p(1,2)}
A2: {a(2,1,3), c(1), c(2), m(1,1), t(2,1), m(2,1), m(2,2), p(2,1), p(1,2)}
A3: {a(2,1,3), c(1), c(2), m(1,1), m(1,2), m(2,1), m(2,2)}
```

Si supponga di aggiungere i seguenti weak constraint al programma P. Si calcoli quale sarebbe il costo di ognuno degli answer set riportati sopra, e si indichi quello ottimo, commentando il procedimento seguito.

```
:~ t(X,Y), not p(X,Y). [X:Y]
:~ m(X,Y), Z=X+Y. [X:Z]
```

b)

A1: 2@1 1@2 2@3 2@4

```
:~ t(2,1), not p(2,1).   [2@1,2,1]
:~ m(1,1), 2=1+1.        [1@2,1,1]
:~ m(2,1), 3=2+1.        [2@3,2,1]
:~ m(2,2), 4=2+2.        [2@4,2,2]
```

A2: 1@2 2@3 2@4 OPTIMUM

```
:~ m(1,1), 2=1+1.        [1@2,1,1]
:~ m(2,1), 3=2+1.        [2@3,2,1]
:~ m(2,2), 4=2+2.        [2@4,2,2]
```

A3: 1@2 3@3 2@4

```
:~ m(1,1), 2=1+1.   [1@2,1,1]
:~ m(1,2), 3=1+2.   [1@3,1,2]
:~ m(2,1), 3=2+1.   [2@3,2,1]
:~ m(2,2), 4=2+2.   [2@4,2,2]
```

L'AS ottimo è A2

Esercizio 2. Siano dati 3 insiemi, finiti, contenenti numeri interi positivi; siano X, Y e Z. Si vogliono selezionare delle triple di numeri interi estratte dall'insieme costituito dal prodotto cartesiano X x Y x Z dei tre insiemi dati, tenendo conto del fatto che i 3 insiemi in input non sono necessariamente disgiunti (cioè, un numero intero può comparire in più di uno degli insiemi), e di quanto espresso di seguito.

- (s1). Considerato un elemento di un insieme, questo non può comparire in più di una tripla (cioè, non esistono due triple distinte che abbiano uno stesso numero nella stessa posizione).
- (s2). È data in input una funzione che calcola quanto si "guadagna" per ciascuna tripla. È necessario selezionare un insieme di triple il cui premio totale (i.e., la somma dei premi di tutte le triple) sia superiore ad una certa soglia, anche questa fornita in input.
- (w1). Si preferiscono triple i cui tre elementi sono tutti diversi.
- (w2). In subordine, se una tripla non è composta da tre elementi tutti diversi tra loro, si preferisce che contenga almeno due elementi diversi (i.e., si preferisce non avere triple composte tutte dallo stesso numero).

Si noti che. tra w1 e w2, è più importante soddisfare w1. Si noti, inoltre, che la preferenza w2 deve essere tenuta in considerazione solo per le triple per le quali non è possibile soddisfare w1.

Modello dei dati in INPUT:

- x(E), y(E), z(E) ← i tre insiemi in input
- reward(E1,E2,E3,R) ← se si sceglie una tripla <E1,E2,E3> si guadagna "R"
- minReward(M) ← il guadagno minimo da ottenere con le triple da selezionare

```
-isTriple(A,B,C) | isTriple(A,B,C) :- x(A), y(B), z(C).

% S1
:- x(A), #count{A,B,C: isTriple(A,B,C)} > 1.
```

```
:- y(B), #count{A,B,C : isTriple(A,B,C)} > 1.
:- z(C), #count{A,B,C : isTriple(A,B,C)} > 1.

% S2
myReward(S) :- S = #sum{R, A,B,C : isTriple(A,B,C), reward(A,B,C,R)}.
:- myReward(A), minReward(B), A < B.

% W1
:~ isTriple(A,A,B), A != B.   [1@2, A,A,B]
:~ isTriple(A,B,A), A != B.   [1@2, A,B,A]
:~ isTriple(B,A,A), A != B.   [1@2, B,A,A]

% W2
:~ isTriple(A,A,A).   [1@1, A,A,A]
```