

Esercizio 1 (6 punti)

Si traducano le seguenti frasi nella logica dei predicati del primo ordine, poi in forma a clausole:

- I libri sono saggi o romanzi (OR semplice)*
- Qualsiasi libro scritto da uno scrittore specializzato in narrativa è un romanzo.*
- Mario è uno scrittore specializzato in storia.*
- Luca è uno scrittore specializzato in narrativa.*
- Luca ha scritto il libro “viaggiare”*

Si usino i seguenti predicati:

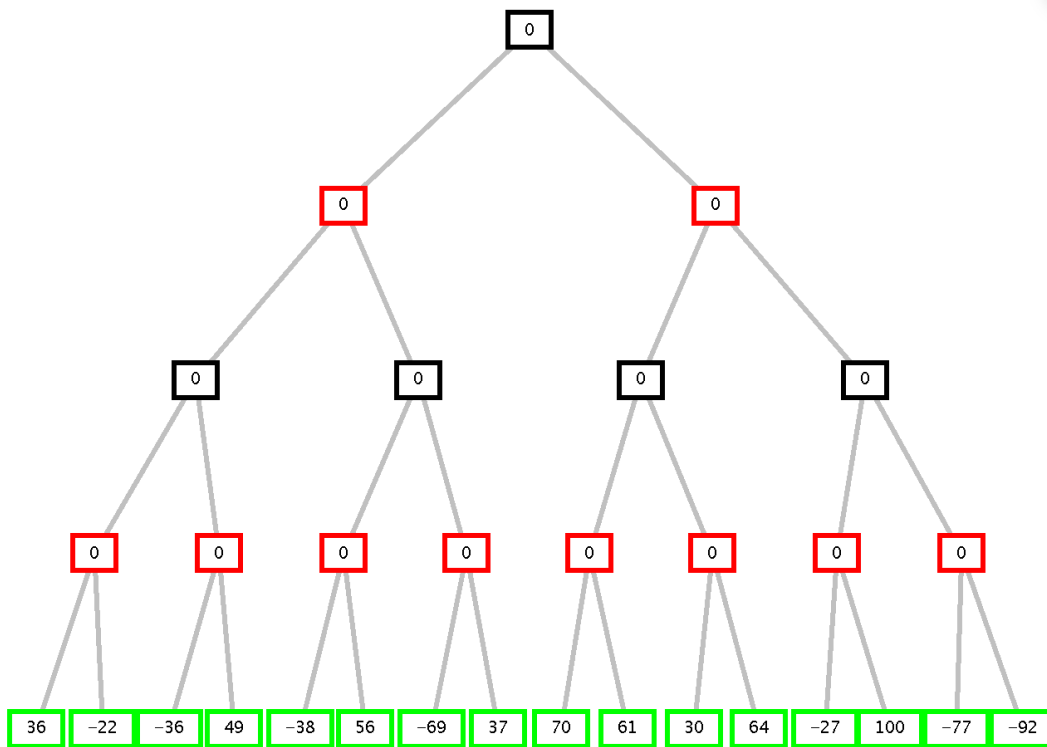
- libro(X)** : X è un libro
- scrittore(X)**: X è uno scrittore
- specialista(X,Y)**: X è specializzato in Y
- saggio(X)**: X è un saggio
- romanzo(X)**; X è un romanzo
- ha_scritto(X,Y)**: X ha scritto Y (ovvero Y è scritto da X).

e le costanti: **luca, mario, viaggiare, narrativa, storia.**

Si usi poi il principio di risoluzione per dimostrare che esiste un libro che è un romanzo e è scritto da Luca.

Esercizio 2 (5 punti)

Si consideri il seguente albero di gioco in cui il primo giocatore è MAX.



- a) Si indichi come l’algoritmo min-max risolve il problema indicando il valore con cui viene etichettato il nodo iniziale e la mossa selezionata dal primo giocatore (arco a sinistra o a destra).
- b) Si mostrino poi i tagli che l’algoritmo alfa-beta consente, indicando gli archi che verranno tagliati.

Esercizio 3 (5 punti)

Si consideri il seguente Programma Prolog:

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

e il goal:

```
?-member(b, [a,b,b]).
```

Disegnare l’albero SLD relativo al goal (tutti rami, sino alle foglie, anche per più soluzioni), e successivamente indicare quali rami sarebbero tagliati nel caso la prima clausola del predicato **member** contenesse un cut:

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

Esercizio 4 (5 punti)

Si scriva in Prolog il predicato `dispariList(L,S)`, che risulta vero quando `S` rappresenta la lista degli elementi di `L` in posizione dispari (prima, terza ecc.)

Esempi

```
?- dispariList([1,5,2,7,9,0],X).
```

```
X = [1, 2, 9]
```

```
?- dispariList([],X).
```

```
X = []
```

```
?- dispariList([1],X).
```

```
X = [1]
```

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

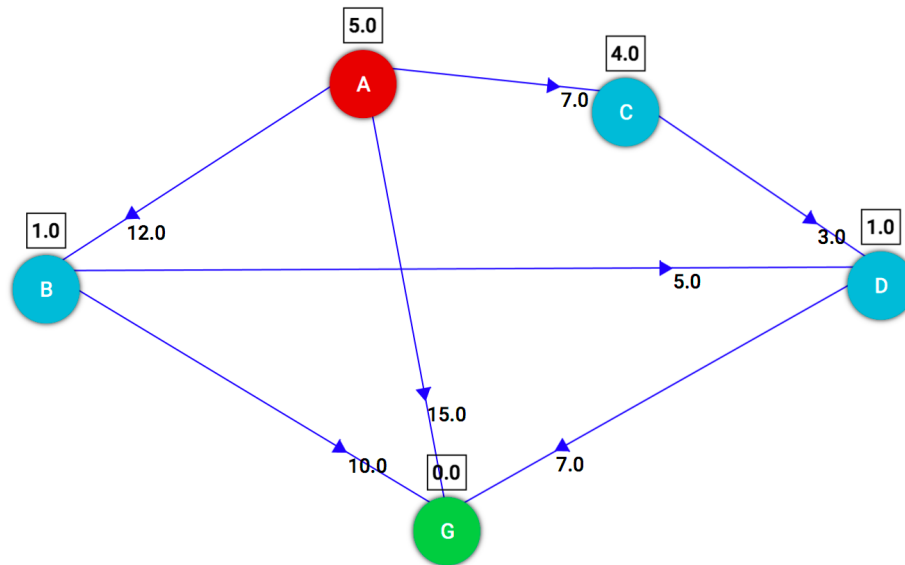
```
true
```

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

```
false
```

Esercizio 5 (7 punti)

Si consideri il seguente grafo, dove A è il nodo iniziale e G il nodo goal, e il numero associato agli archi è il costo dell'operatore per andare dal nodo di partenza al nodo di arrivo dell'arco. Vicino ad ogni nodo, in un quadrato, è indicata inoltre la stima euristica della sua distanza dal nodo goal G:



Si applichi la ricerca **depth-first** su alberi (non tenendo quindi traccia dei nodi già visitati) **disegnando l'albero generato dinamicamente** e indicando:

- i nodi espansi nell'ordine di espansione;
- i nodi sulla strada della soluzione e il costo della soluzione;
- se è garantita o meno l'ottimalità.

Si mostri poi cosa succederebbe invece con ricerca **Greedy-Best-First**.

Esercizio 6 (4 punti)

Si spieghi brevemente il predicato PROLOG predefinito `findall/3`.

Poi si consideri il seguente programma PROLOG:

```
child(martha,charlotte).
child(charlotte,caroline).
child(caroline,laura).
child(laura,rose).
descend(X,Y) :- child(X,Y).
descend(X,Y) :- child(X,Z), descend(Z,Y).
```

Cosa risponderebbe PROLOG alle seguenti query?

```
?-findall(X,descend(martha,X),Z).
```

```
?-findall(X,descend(mary,X),Z).
```

Cosa succederebbe nel caso di utilizzo del predicato `bagof/3` al posto di `findall/3` nelle query precedenti?

7 Luglio 2022 - Soluzioni

Esercizio 1

Formule in logica del I ordine:

$\forall X \text{ libro}(X) \rightarrow \text{saggio}(X) \text{ or } \text{romanzo}(X)$ (OR semplice)

$\forall Y \forall X \text{ scrittore}(X) \text{ and } \text{specialista}(X, \text{narrativa}) \text{ and } \text{libro}(Y) \text{ and } \text{ha_scritto}(X, Y) \rightarrow \text{romanzo}(Y)$

$\text{scrittore}(\text{mario}) \text{ and } \text{specialista}(\text{mario}, \text{storia})$.

$\text{scrittore}(\text{luca}) \text{ and } \text{specialista}(\text{luca}, \text{narrativa})$.

$\text{libro}(\text{viaggiare}) \text{ and } \text{ha_scritto}(\text{luca}, \text{viaggiare})$

Goal: $\exists X \text{ libro}(X) \text{ and } \text{ha_scritto}(\text{luca}, X) \text{ and } \text{romanzo}(X)$

Trasformazione in clausole:

C1: $\text{not libro}(X) \text{ or } \text{saggio}(X) \text{ or } \text{romanzo}(X)$

C2: $\text{not scrittore}(X) \text{ or } \text{not specialista}(X, \text{narrativa}) \text{ or } \text{not libro}(Y) \text{ or } \text{not ha_scritto}(X, Y) \text{ or } \text{romanzo}(Y)$

C3: $\text{scrittore}(\text{mario})$

C4: $\text{specialista}(\text{mario}, \text{storia})$

C5: $\text{scrittore}(\text{luca})$

C6: $\text{specialista}(\text{luca}, \text{narrativa})$

C7: $\text{libro}(\text{viaggiare})$

C8: $\text{ha_scritto}(\text{luca}, \text{viaggiare})$

Goal negato e trasformato in clausola:

GN: $\text{not libro}(X) \text{ or } \text{not ha_scritto}(\text{luca}, X) \text{ or } \text{not romanzo}(X)$

Risoluzione:

C2 + C5 = C9: $\text{not specialista}(\text{luca}, \text{narrativa}) \text{ or } \text{not libro}(Y) \text{ or } \text{not ha_scritto}(\text{luca}, Y) \text{ or } \text{romanzo}(Y)$

C9 + C6 = C10: $\text{not libro}(Y) \text{ or } \text{not ha_scritto}(\text{luca}, Y) \text{ or } \text{romanzo}(Y)$

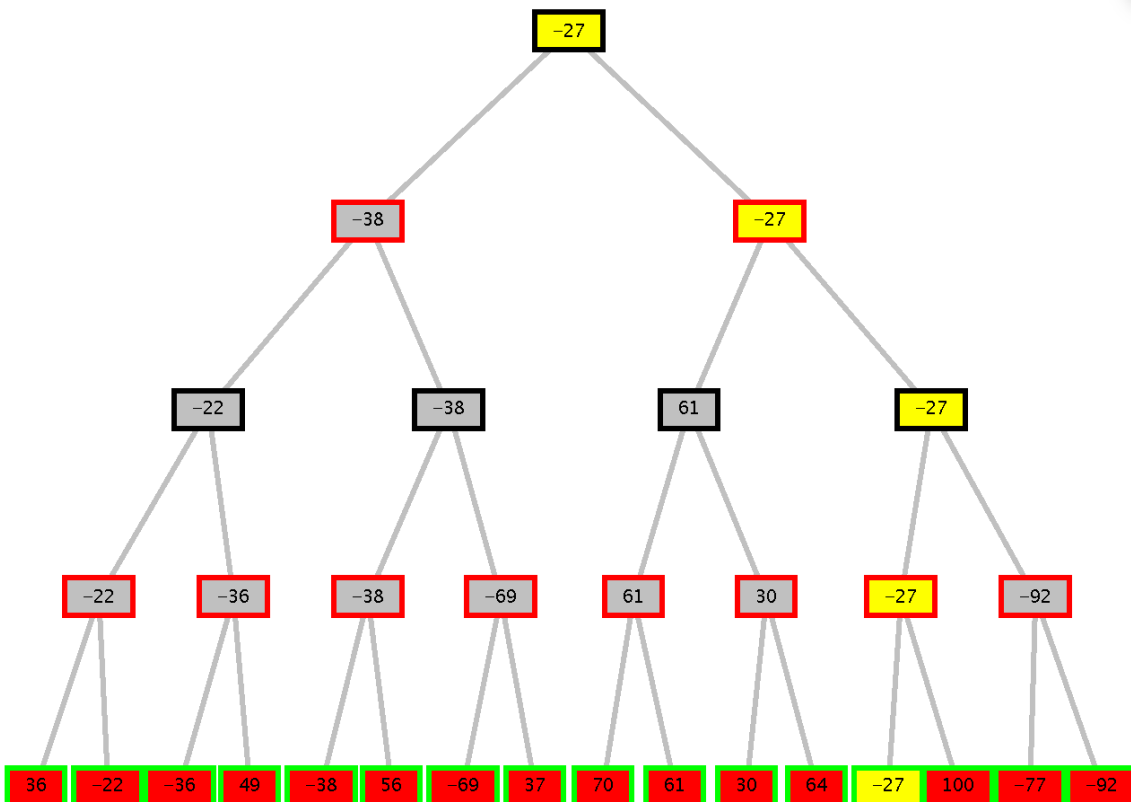
C10 + GN = C11: $\text{not libro}(Y) \text{ or } \text{not ha_scritto}(\text{luca}, Y)$

C11 + C7 = C12: $\text{not ha_scritto}(\text{luca}, \text{viaggiare})$

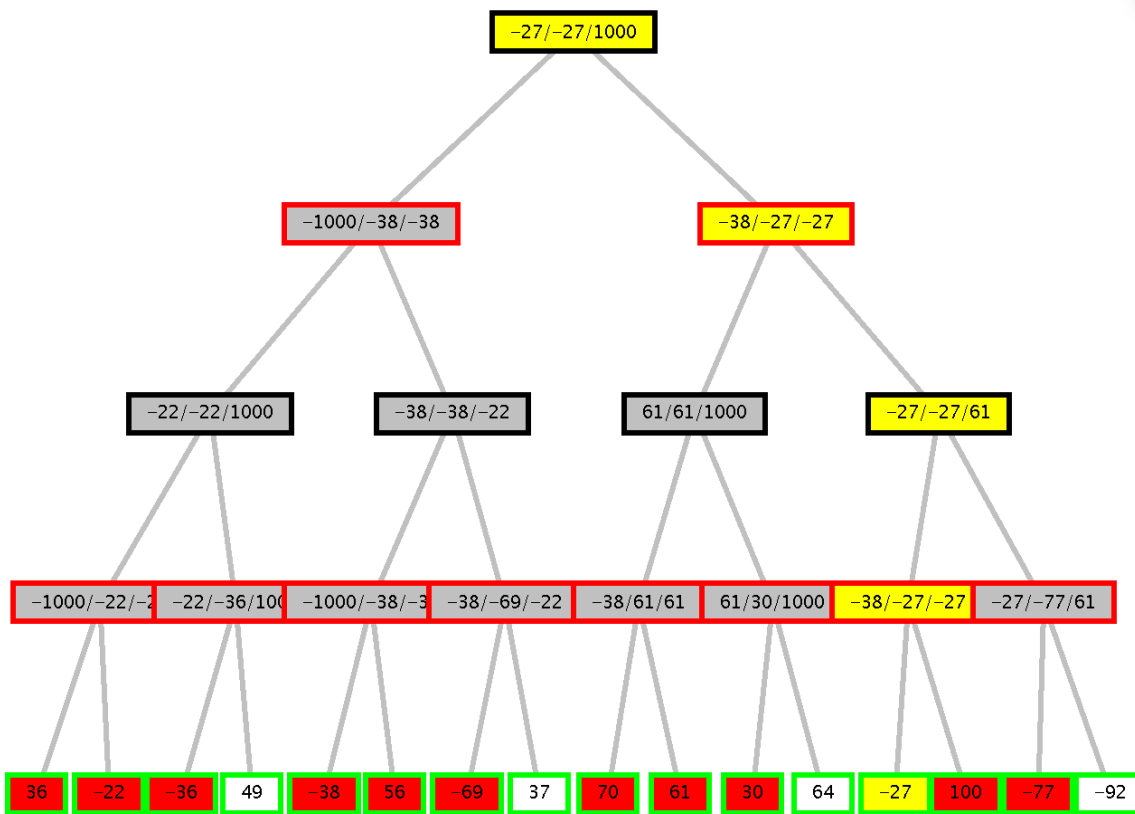
C12 + C8 = clausola vuota

Esercizio 2

Min-max: strada in giallo, valore nodo radice -27, ramo a2.



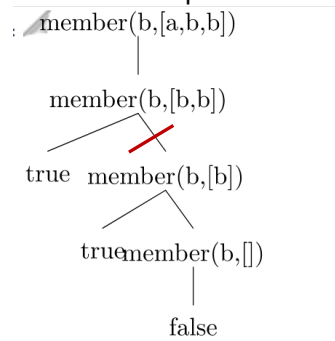
Alfa-Beta: In rosso i nodi espansi, in giallo la strada trovata, i nodi in bianco non sono esplorati per effetto dei tagli alfa-beta.



Archi tagliati a18,a22, a26, a30 (4 tagli). Scelta per il ramo a2, valore propagato -27.

Esercizio 3

In rosso il ramo tagliato per effetto del cut introdotto nella prima clausola di member.

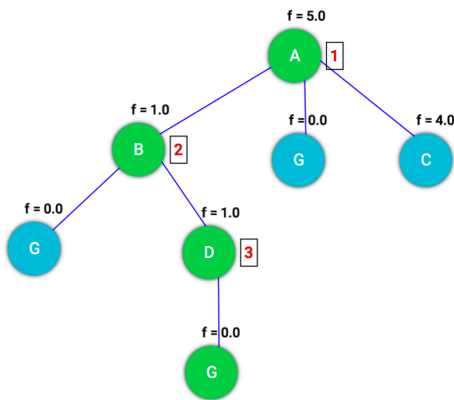


Esercizio 4

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

Esercizio 5

Con Depth First i nodi espansi sono ABDG (nodi sul cammino soluzione) e la soluzione ha costo 24 (non ottimale):



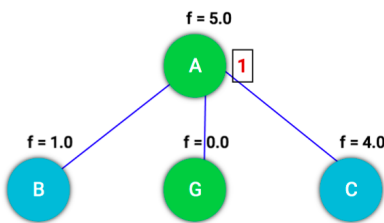
Operations

Show operations

- 1) A [f = 5.0]
- 2) B [f = 1.0]
- 3) D [f = 1.0]
- /) G [f = 0.0]
- /) G [f = 0.0]
- /) G [f = 0.0]
- /) C [f = 4.0]

Path cost: 24.0
 Nodes expanded: 3
 Queue size: 3
 Max queue size: 4

Con Best First, i nodi espansi sono AG, la soluzione trovata AG ha costo 15:



Operations

Show operations

- 1) A [f = 5.0]
- /) B [f = 1.0]
- /) G [f = 0.0]
- /) C [f = 4.0]

Path cost: 15.0
 Nodes expanded: 1
 Queue size: 2
 Max queue size: 3

Esercizio 6

Per la descrizione del predicato findall /3 si consulti il materiale del corso.

?- findall(X,descend(martha,X),Z).

Produce una lista Z che contiene tutti i valori di X che soddisfano descend(martha,X) .

La risposta sarà:

findall(X,descend(martha,X),Z).

Z = [charlotte, caroline, laura, rose]

Con la seguente query:

?- findall(X,descend(mary,X),Z).

Z = []

Non ci sono soluzioni infatti per il goal descend(mary,X) e findall/3 ritorna la lista vuota.

bagof/3 darebbe la stessa risposta di findall/ 3 alla prima query, ma fallirebbe nella seconda in cui non ci sono soluzioni.

?- bagof(X,descend(martha,X),Z).

Z = [charlotte, caroline, laura, rose]

?-bagof(X,descend(mary,X),Z).

false