FONDAMENTI DI INTELLIGENZA ARTIFICIALE

7 Luglio 2022 – Tempo a disposizione: 2 h – Risultato: 32/32 punti

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:

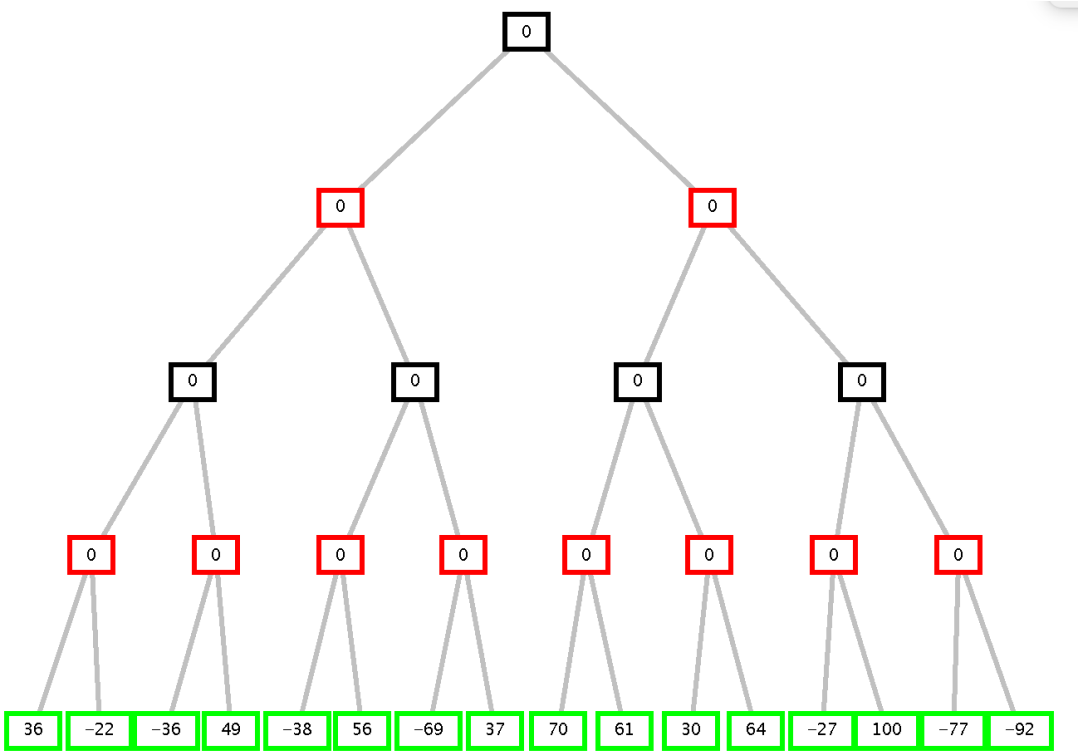
libro(X) : X è un libro
scrittore(X): X è uno scrittore
specialista(X,Y): X è specializzato in Y
saggio(X): X è un saggio
romanzo(X); X è un romanzo
ha_scritto(X,Y): X ha scritto Y (ovvero Y è scritto da X).

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

e il goal:

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

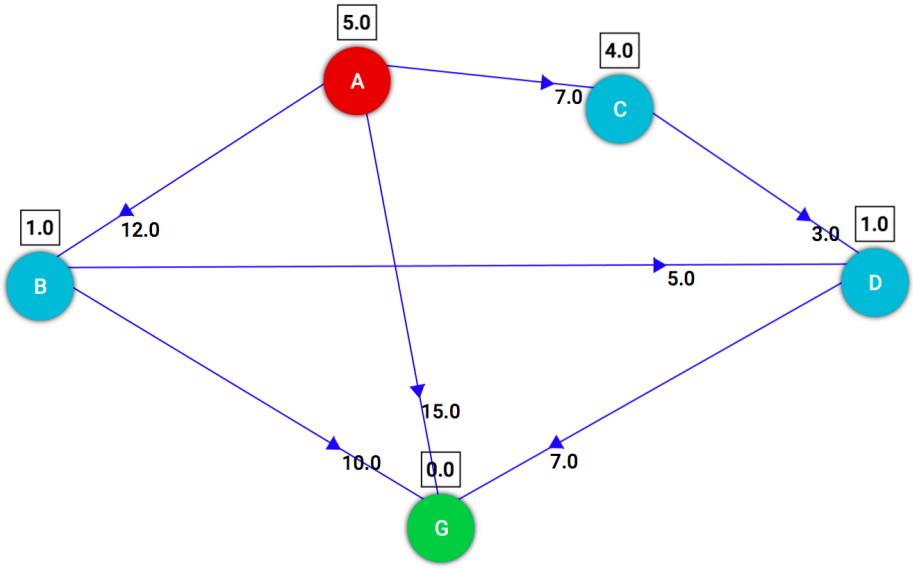
Disegnare l'albero SLD relativo al goal (tutti rami, sino alle foglie, anche per più soluzioni), e successivamenteindicare quali rami sarebbero tagliati nel caso la prima clausola del predicato **member** contenesse un cut:

```
member(X,[X|_]):-!.
member(X,[Y|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.** In caso di non determinismo i nodi si selezionino per l'espansione in ordine alfabetico.
Si indichino:

- 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:

∀X libro(X) -> saggio(X) or romanzo(X) (OR semplice)
∀Y ∀X scrittore(X) and specialista(X,narrativa) and libro(Y) and ha_scritto(X,Y) -> romanzo(Y)
scrittore(mario) and specialista(mario,storia).
scrittore(luca) and specialista(luca,narrativa).
libro(viaggiare) and ha_scritto(luca, viaggiare)

Goal: ∃X libro(X) and ha_scritto(luca,X) and romanzo(X)

Trasformazione in clausole:

C1: not libro(X) or saggio(X) or romanzo(X)
C2: not scrittore(X) or not specialista(X,narrativa) or not libro(Y) or not ha_scritto(X,Y) or romanzo(Y)
C3: scrittore(mario)
C4: specialista(mario,storia)
C5: scrittore(luca)
C6: specialista(luca, narrativa)
C7: libro(viaggiare)
C8: ha_scritto(luca,viaggiare)

Goal negato e trasformato in clausola:

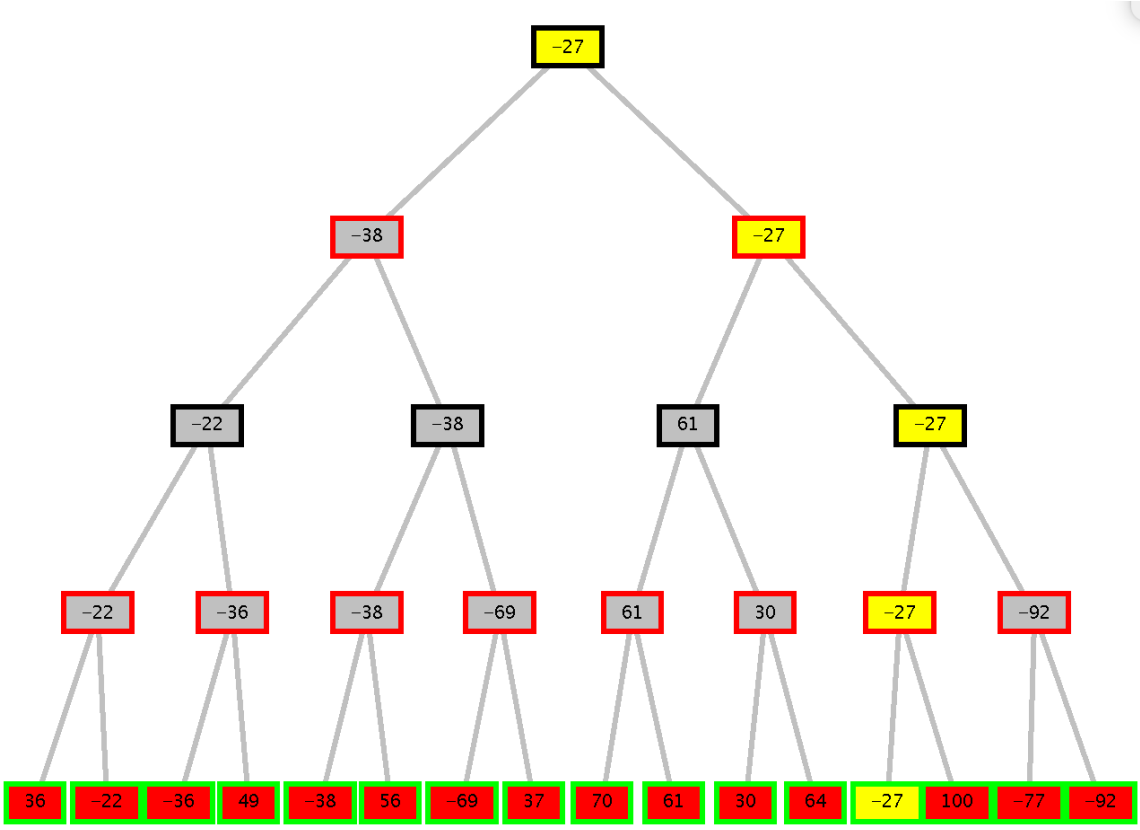
GN: not libro(X) or not ha_scritto(luca,X) or not romanzo(X)

Risoluzione:

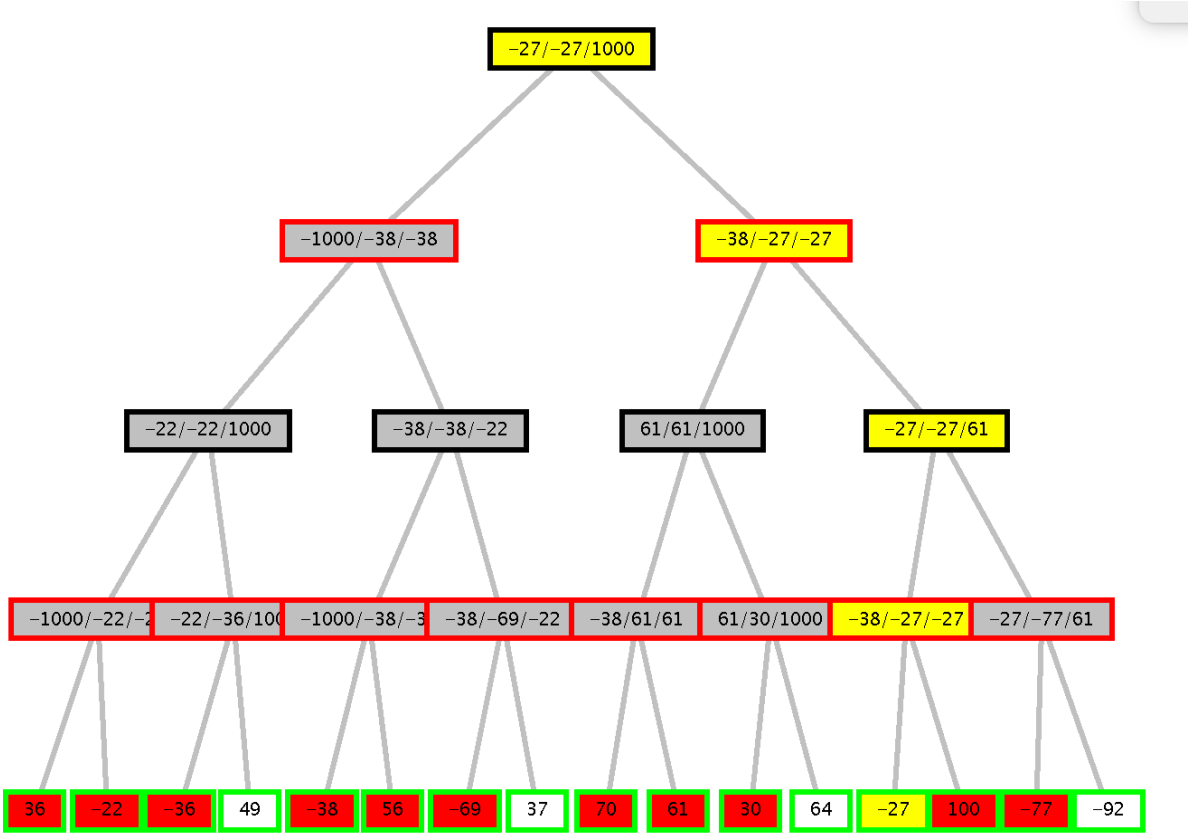
C2 + C5= C9: not specialista(luca,narrativa) or not libro(Y) or not ha_scritto(luca,Y) or romanzo(Y)
C9+C6=C10: not libro(Y) or not ha_scritto(luca,Y) or romanzo(Y)
C10+GN=C11 not libro(Y) or not ha_scritto(luca,Y)
C11+C7=C12 not ha_scritto(luca,viaggiare)
C12+C8= clausola vuota

Esercizio 2

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



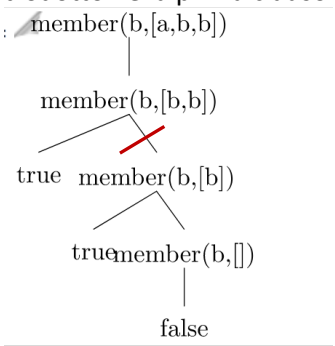
Alfa-Beta: In rosso i nodi espansi, in giallo la strada trovata, i nodi in bianco non sono esplorati per effetto dei tagli alfa-beta.



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.

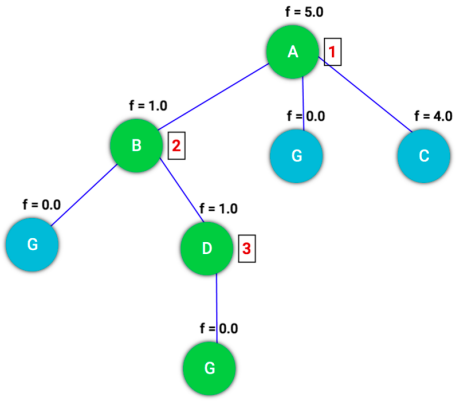


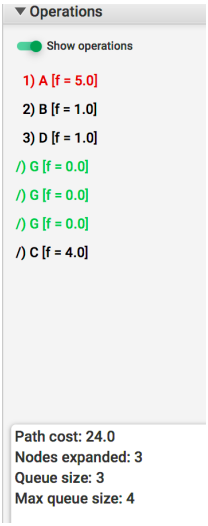
Esercizio 4

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

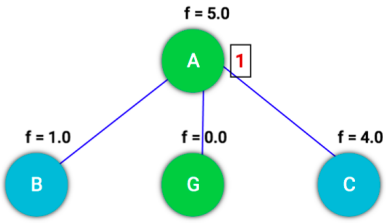
Esercizio 5

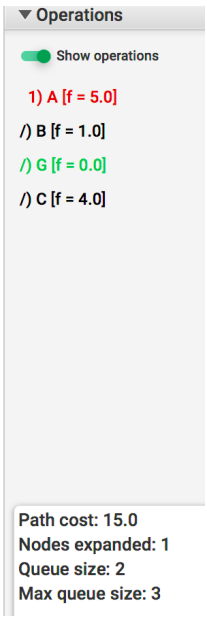
Con Depth First i nodi espansi sono ABDG (nodi sul cammino soluzione) e la soluzione ha costo 24 (non ottimale):





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





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