FONDAMENTI DI INTELLIGENZA ARTIFICIALE

11 Gennaio 2024 - Tempo a disposizione: 2 h – Risultato: 32/32 punti

Esercizio 1 (7 punti)

Si formalizzino le seguenti frasi in logica dei predicati del I ordine:

1. *tutti i blocchi sul tavolo sono bianchi o (XOR) neri;*
2. *tutti i blocchi bianchi sono a forma di cubo;*
3. *tutti i blocchi neri sono a forma di cono;*
4. *Il blocco b1 è sul tavolo;*
5. *Il blocco b1 non è nero.*

Le si trasformi in clausole e si usi la risoluzione per dimostrare che:

Esiste un blocco a forma di cubo.

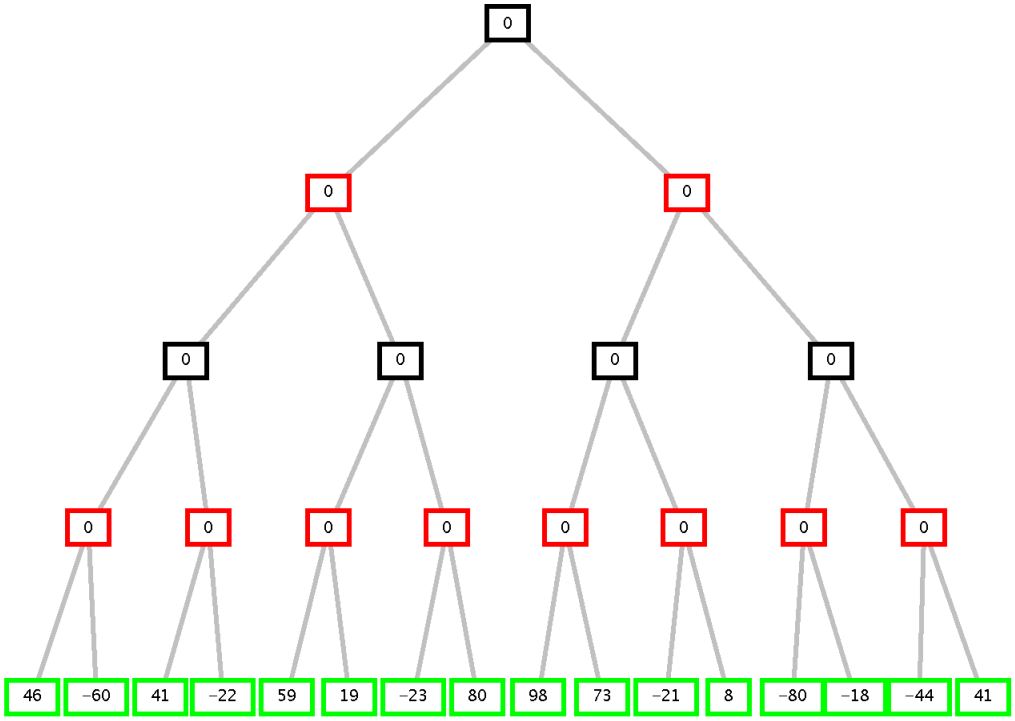
Si utilizzino I seguenti predicati:

blocco(X): X è un blocco; *sul_tavolo(X)*: X è sul tavolo;
nero(X): X è nero; *bianco(X)*: X è bianco;
cubo(X): X è a forma di cubo; *cono(X)*: X è a forma di cono.

e la costante *b1.*

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 scriva un predicato Prolog **select(L1, M, N, L2, L3)** che data una lista di interi **L1** e due numeri interi **N** e **M**, con **N>=M**, crei due liste, inserendo nella lista **L2** i numeri maggiori di M e (AND) minori di N, e nella lista **L3** i numeri maggiori di N o (OR) minori di M. I numeri uguali a M e N non devono essere inseriti in alcuna delle due liste.

Esempi:

?-select([1,2,7,3,4,5,6,9,6,9], 4, 7, [5,6,6], [1,2,3,9,9]).
Yes
?-select([1,2,7,3,4,5,6,9,6,9], 4, 7, A, B).
Yes A = [5,6,6], B = [1,2,3,9,9]

Esercizio 4 (6 punti)

Dato il seguente programma Prolog **`includeList(L1,L2,L3)`**:

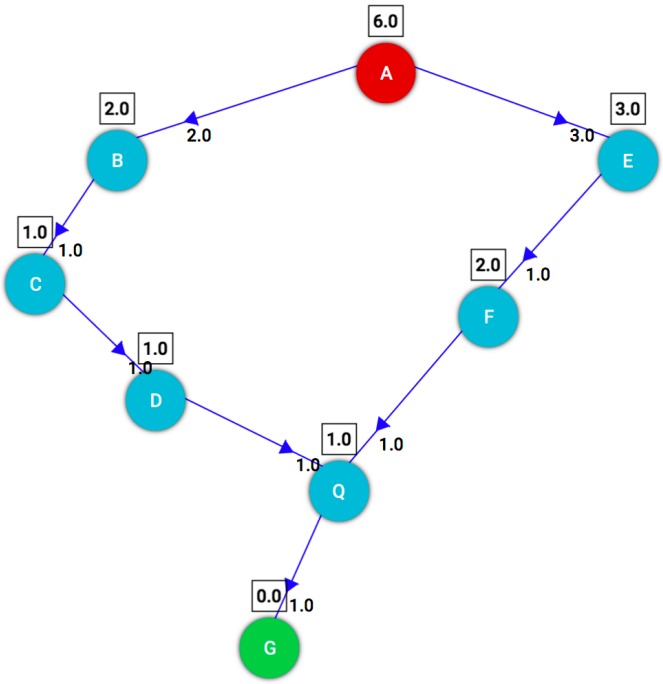
```
includeList([],X,X):-!.
includeList([X|A], [X|B], Y) :- !, includeList(A,B,Y).
includeList([X|A], B, [X|Y]) :- includeList(A,B,Y).
```

che, date tre liste di interi **`L1,L2`** e **`L3`** ha successo se la lista **`L2`** è esattamente inclusa in **`L1`** con i valori nello stesso ordine o se **`L2`** risulta vuota**.** Per ipotesi, la lista **`L2`** non è mai più lunga della lista **`L1`**. Inoltre, **`L3`** deve contenere gli elementi di **`L1`** non presenti in **`L2`** o fuori ordine rispetto a quelli di L1.

Si mostri l'*albero SLD* relativo al goal: **`?-includeList([0,1,3,5],[1,3],X)`**. indicando i rami di fallimento, di successo e quelli tagliati dal cut. Si indichi anche la risposta calcolata per la variabile **`X`** del goal.

Esercizio 5 (5 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 **A*** su alberi (non tenendo quindi traccia dei nodi già visitati che non vengono automaticamente eliminati) **disegnando l'albero generato dinamicamente.** In caso di non determinismo si selezionino i nodi da espandere secondo l'ordine alfabetico. Si indichino:

- i nodi espansi nell'ordine di espansione;
- i nodi sulla strada della soluzione e il costo della soluzione;
- la condizione sulla stima euristica h(n) che garantisce l'ottimalità della ricerca su alberi e se è soddisfatta in questo caso.

Esercizio 6 (4 punti)

Si disegni il constraint graph per il seguente problema CSP:

A::[1,2, 3, 4]	C::[1,2, 3, 4]
B::[1,2, 3, 4]	D::[1,2, 3, 4]
A>B	A<D
C<D	A!=C
B!=C	

e, dopo avere brevemente introdotto l'algoritmo AC3 (Arc Consistency 3), se ne mostri l'esecuzione sull'esempio.

11 Gennaio 2024 - Soluzioni

Esercizio 1

Rappresentazione in FOL:

∀ X blocco(X) and sul_tavolo(X) -> (bianco(X) ex-or nero(X)).
∀ X blocco(X) and bianco(X) -> cubo(X).
∀ X blocco(X) and nero(X) -> cono(X).
blocco(b1) and sul_tavolo(b1).
blocco(b1) and not nero(b1).

Query: ∃X (blocco(X) and cubo(X)).

Conversione in clausole:

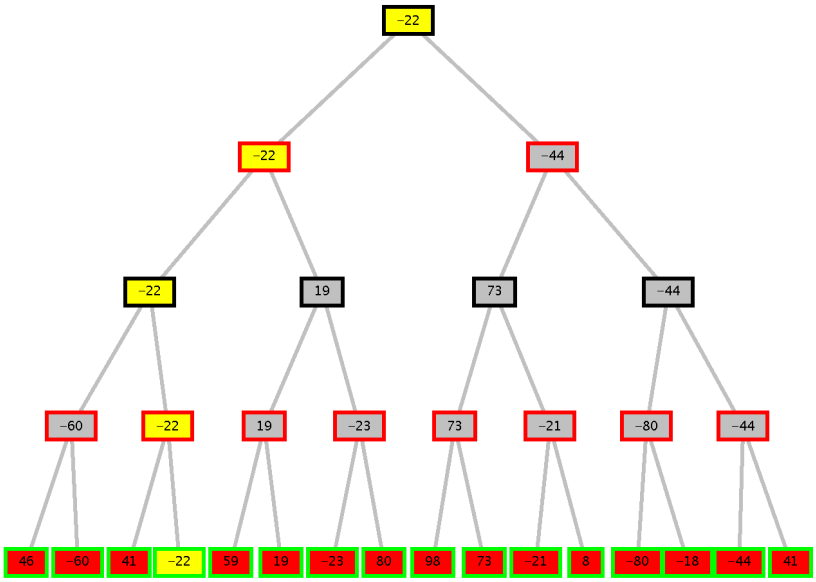
C1a: not blocco(X) or not sul_tavolo(X) or nero(X) or bianco(X)
C1b: not blocco(X) or not sul_tavolo(X) or not nero(X) or not bianco(X)
C2: not blocco(X) or not bianco(X) or cubo(X).
C3: not blocco(X) or not nero(X) or cono(X).
C4a: blocco(b1)
C4b: sul_tavolo(b1).
C5a: blocco(b1).
C5b: not nero(b1).
Query NEGATA QN: not blocco(X) or not cubo(X).

Risoluzione:

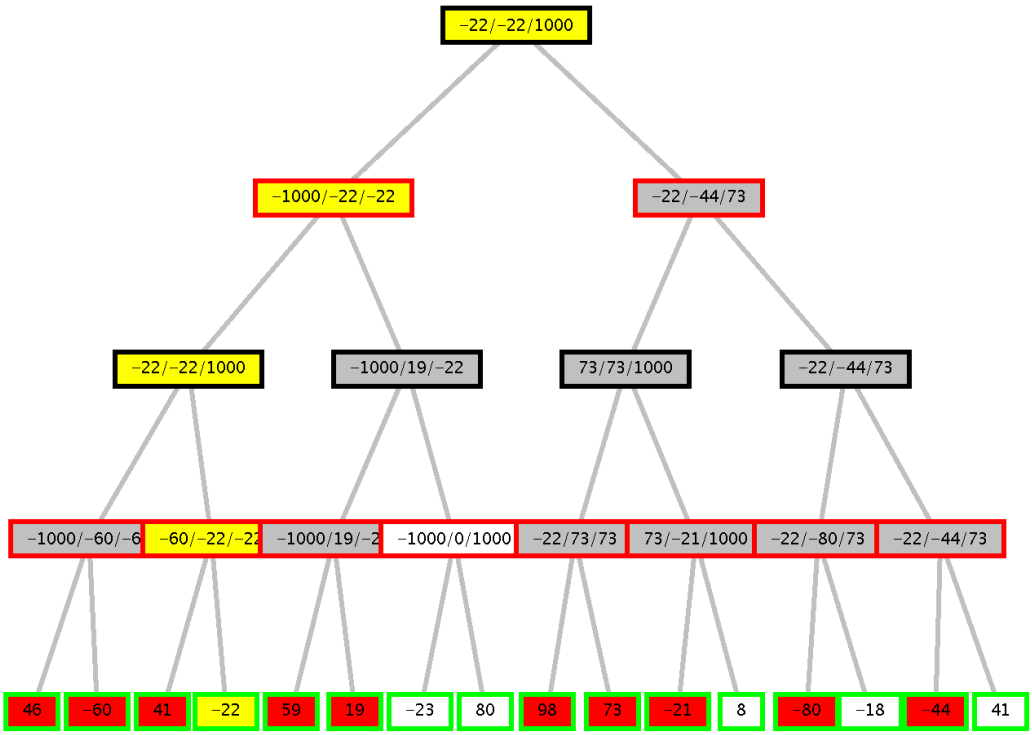
QN + C4a = C6: not cubo(b1).
C6 + C2 = C7 not blocco(b1) or not bianco(b1).
C7 + C4a= C8 not bianco(b1).
C8 + C1a= C9 not blocco(b1) or not sul_tavolo(b1) or nero(b1).
C9 + C4a = C10 not sul_tavolo(b1) or nero(b1).
C10 + C4b = C11 nero(b1).
C11 + C5b= contraddizione! Dimostrato
Oppure:
QN + C2 = C6: not blocco(X) or not bianco(X).
C6 + C1a = C7: not blocco(X) or not sul_tavolo(X) or nero(X).
C7 + C5b = C8: not blocco(b1) or not sul_tavolo(b1).
C8 + C4b = C9: not blocco(b1).
C9 + C4a = contraddizione.

Esercizio 2

Min max:



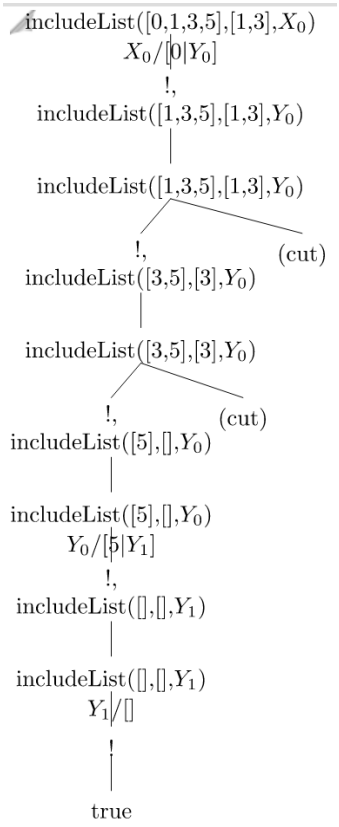
Alfa Beta:



Esercizio 3

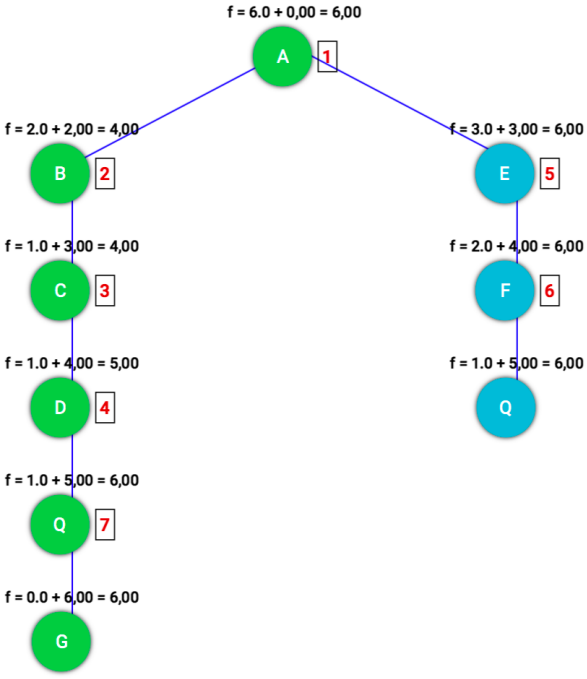
```
select([], _, _, [], []).
select([A|R], M, N, [A|Ra], Rb) :-  A<N,A>M, !, select(R,M,N, Ra,Rb).
select([A|R], M, N, Ra, [A |Rb] ) :-  A<M, !, select(R,M,N, Ra,Rb).
select([A|R], M, N, Ra, [A |Rb] ) :-  A>N, !, select(R,M,N, Ra,Rb).
select([_|R],M, N, Ra ,Rb ) :-   select(R,M, N, Ra,Rb).
```

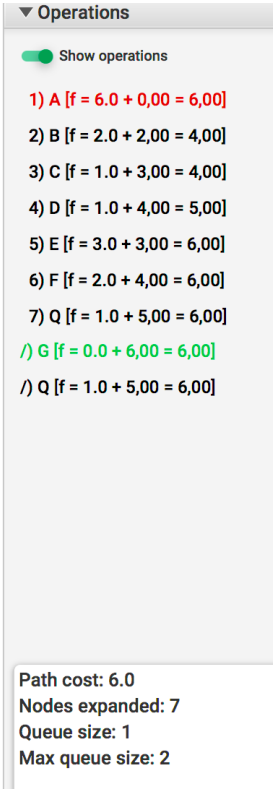
Esercizio 4



Esercizio 5

Con A* i nodi espansi sono ABCDEFQG e la soluzione ha costo 6 (ottimale):





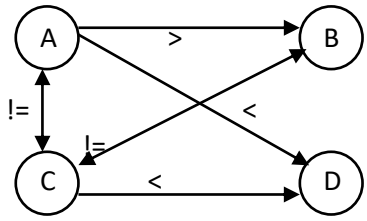
La condizione sulla funzione euristica stimata h* (n) che garantisce l'ottimalità della ricerca è la condizione di ammissibilità che deve valere per ogni nodo dell'albero e che è verificata se la h*(n) è ottimista cioè h*(n) <= h(n). Tale condizione è soddisfatta in questo caso.

Esercizio 6

A::[1,2, 3, 4] C::[1,2, 3, 4]

B::[1,2, 3, 4]
A>B
C<D
B!=C

D::[1,2, 3, 4]
A<D
A!=C



	A	**B**	**C**	**D**
1 iterazione	[1,2, 3, 4]	[1,2, 3, 4]	[1,2, 3, 4]	[1,2, 3, 4]
A>B e B<A	[2,3,4]	[1,2,3]		
A<D e D>A	[2,3]			[3,4]
C<D e D>C			[1,2,3]	[3,4]
C!=A e C!=B				
2 iterazione	[2,3]	[1,2,3]	[1,2,3]	[3,4]
A>B e B<A	[2,3]	[1,2]		
A<D e D>A	[2,3]			[3,4]
C<D e D>C			[1,2,3]	[3,4]
C!=A e C!=B		[1,2]	[1,2,3]	
3 iterazione	[2,3]	[1,2]	[1,2,3]	[3,4]
A>B e B<A	[2,3]	[1,2]		
A<D e D>A	[2,3]			[3,4]
C<D e D>C			[1,2,3]	[3,4]
C!=A e C!=B	[2,3]	[1,2]	[1,2,3]	[3,4]

Quiescenza alla terza iterazione