FONDAMENTI DI INTELLIGENZA ARTIFICIALE

6 Luglio 2023– Tempo a disposizione: 2 h – Risultato: 32/32 punti

Esercizio 1 (6 punti)

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

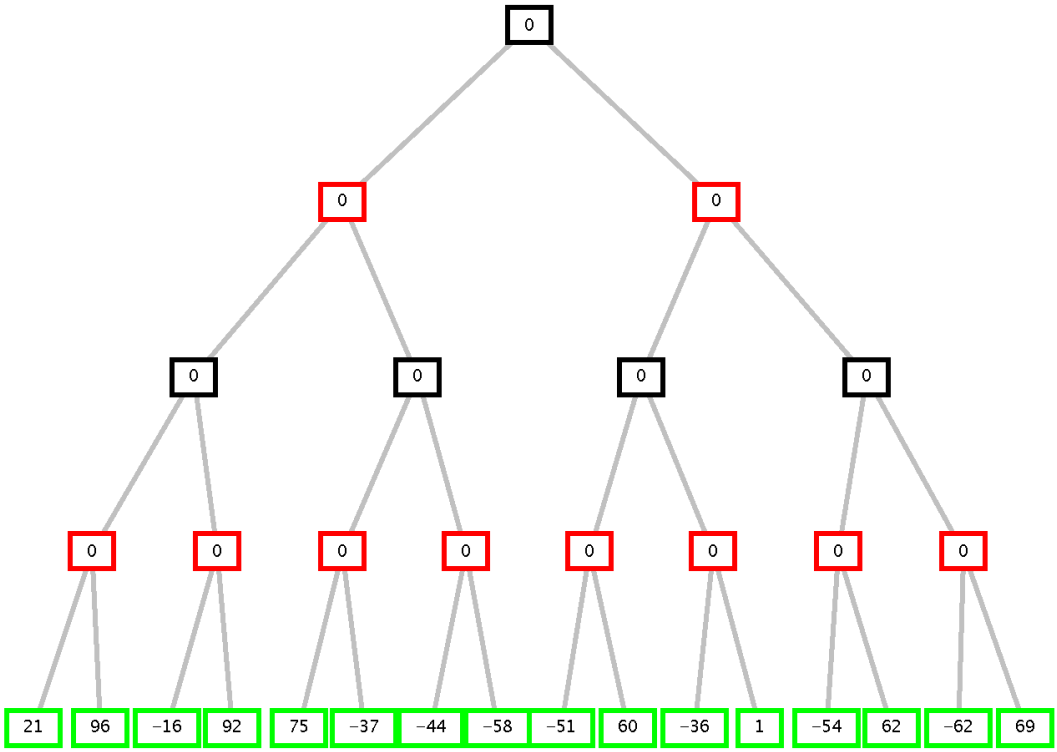
- Ogni persona è o felice o triste (or esclusivo XOR)
- Se una persona è felice è allegra
- Chi è allegro, sorride
- Esiste una persona felice

b) Si usi poi il principio di risoluzione per dimostrare che esiste una persona sorridente e non triste.

Si utilizzino i seguenti predicati con l'ovvio significato: persona(X), felice(X), triste(X), allegra(X), sorride(X)

Esercizio 2 (6 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)

Data una lista di liste di interi **L1**, un numero intero **Soglia** e un numero intero **Find**, si realizzi un predicato PROLOG:

occorrenze(L1, Soglia, Find, L2)

che restituisca una nuova lista di liste **L2** contenente gli elementi di **L1** selezionati in relazione al valore **Soglia** e al numero **Find**; in particolare la lista restituita **L2** deve contenere le liste appartenenti alla lista **L1** solo se contengono il numero **Find** almeno **Soglia** volte. Ad esempio, se L1 = [[2,5,3], [4,5,6,5,5], [1,2,3]], []], Soglia è 2 e Find è 5, il predicato occorrenze deve restituire la lista L2 = [[4,5,6,5,5]].

A tale scopo si realizzi e utilizzi anche il predicato

count(L,Find,Res)

che data una lista L e un numero **Find** restituisca in **Res** il numero di volte in cui **Find** compare in **L**.

Esempio:

?- occorrenze([[2,5,3], [4,5,6,5,5], [1,2,3], []], 2, 5, L2)

L2 = [[4, 5, 6, 5, 5]]

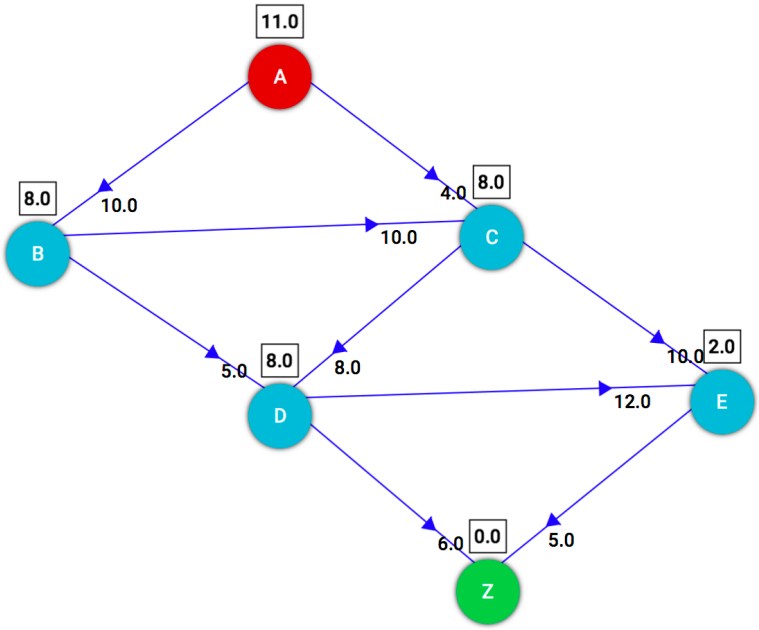
?- occorrenze([], 2, 5, L2)
L2 = []
?- occorrenze([[2,5,3], [4,5,6,5,5], [1,2,3], []], 5, 2, L2)
L2 = []

Esercizio 4 (6 punti)
Il seguente programma Prolog determina i numeri positivi pari (**even**) o dispari (**odd**). Il programma è così definito (aggiungendo anche il predicato **member**):

```
even(0).
even(X):-X>0, X1 is X-1, odd(X1).
odd(1).
odd(X):-X>0, X1 is X-1, even(X1).

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

Si mostri l'*albero SLD* relativo al goal: **?- member(X,[1]), not even(X).**

Esercizio 5 (5 punti)
Si consideri il seguente grafo, dove A è il nodo iniziale e Z 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 quadratino, è indicata inoltre la stima euristica della sua distanza dal nodo goal:



Si applichi la ricerca **Best First** su alberi (non tenendo quindi traccia dei nodi già visitati) **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;
- se la soluzione ottenuta è ottima o no motivando la risposta data.

Esercizio 6 (4 punti)
Si definisca formalmente il concetto di conseguenza logica e di correttezza e completezza di un sistema logico. Si discutano poi le proprietà della Programmazione Logica relativamente a correttezza e completezza motivandole opportunamente.

6 Luglio 2023 – Soluzioni
Esercizio 1

In logica:

- Ogni persona è o felice o triste
  ∀X persona(X) -> felice(X) ex-or triste(X) (OR esclusivo)
- Se una persona è felice è allegra
  ∀X persona(X) and felice(X) -> allegra(X)
- Chi è allegro, sorride
  ∀X allegra(X) -> sorride(X)
- Esiste una persona felice
  ∃Y persona(Y) and felice (Y).

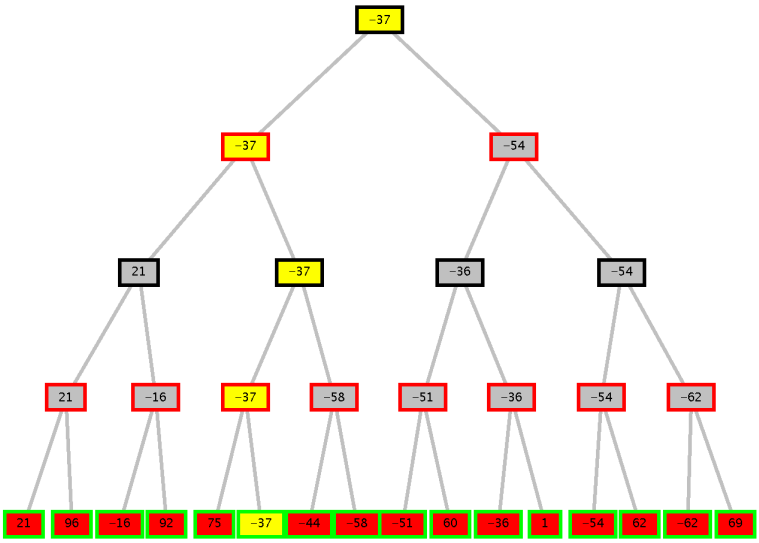
Query: ∃Y persona(Y) and sorride(Y) and not triste(Y).

Trasformazione in clausole:
C1: not persona(X) or felice(X) or triste(X)
C2: not persona(X) or not felice(X) or not triste(X)
C3: not persona(X) or not felice(X) or allegra(X)
C4: not allegra(X) or sorride(X)
C5: persona(c1)
C6: felice(c1)
GoalNeg not persona(Y) or not sorride(Y) or triste(Y).

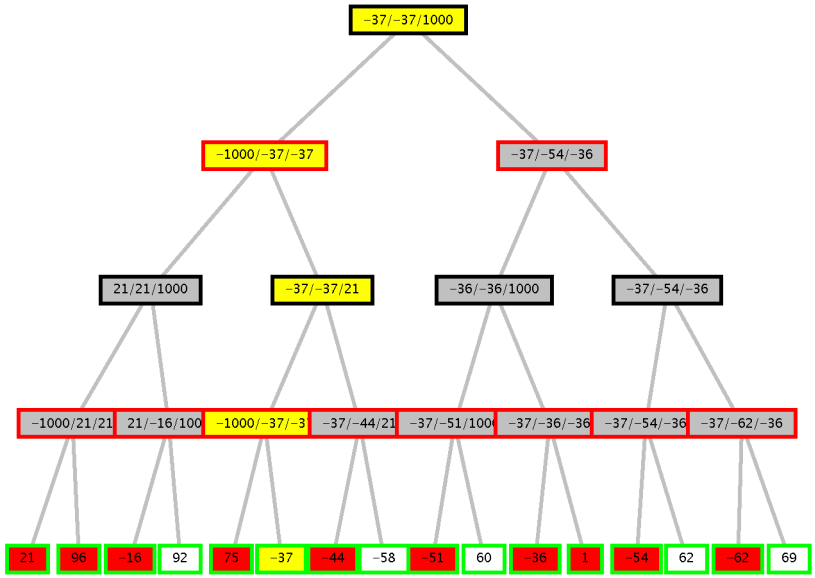
Applicando il Principio di Risoluzione:
C7 (da C5 e GoalNeg): not sorride(c1) or triste (c1).
C8 (da C2 e C5): not felice(c1) or not triste(c1)
C9 (da C8 e C6): not triste (c1)
C10 (da C7 e C9): not sorride(c1)
C11 (da C10 e C4): not allegra(c1)
C12 (da C3 e C11): not persona (c1) or not felice (c1).
C13 (da C12 e C5): not felice (c1)
C14 (da C13 e C6): clausola vuota contraddizione logica

Oppure:
C7 (GoalNeg+C2): not persona(X) or not sorride(X) or not felice(X).
C8 (da C7 e C4): not persona(X) or not felice(X) or not allegra(X).
C9 (da C8 e C3): not persona(X) or not felice(X)
C10 (da C9 e C5): not felice(c1)
C11 (da C10 e C6): clausola vuota contraddizione logica

Esercizio 2
Min-Max: in giallo la strada selezionata.



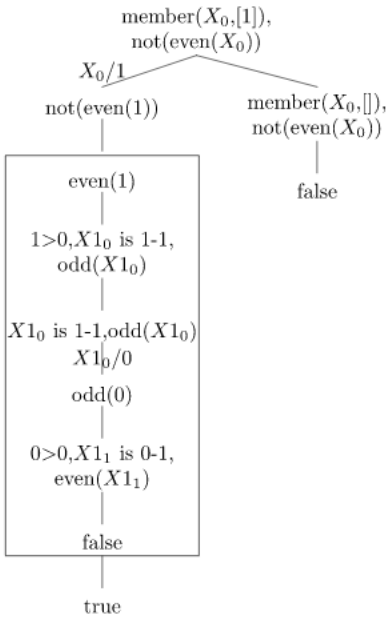
Alfa-Beta: In rosso i nodi espansi, in giallo la strada trovata, i nodi in bianco non sono esplorati per effetto dei tagli alfa-beta.



Esercizio 3

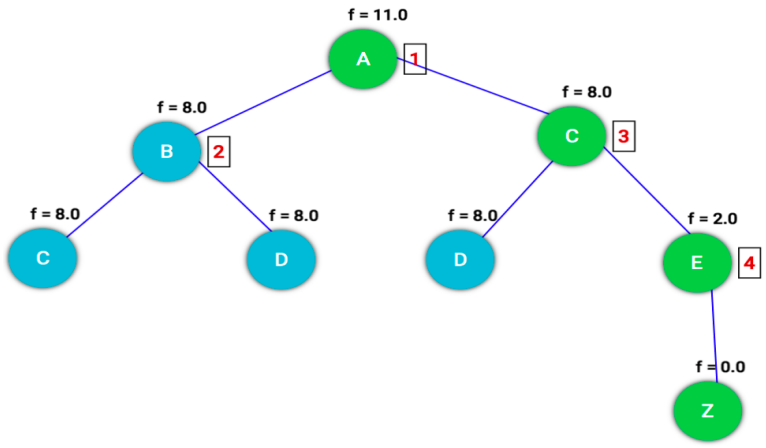
```
occorrenze([], _ , _, []).

occorrenze([X|Y], Soglia , Find, [X|Y1]):-
   count(X,Find,Res),
   Res >= Soglia,
   !,
   occorrenze(Y,Soglia,Find,Y1).
occorrenze([_|Y], Soglia , Find, Y1):-
   occorrenze(Y,Soglia,Find,Y1).

count([],_,0).
count([X|Y],X,N):- !, count(Y,X,N1),  N is N1 +1.
count([_|Y],X,N):- count(Y,X,N).
```

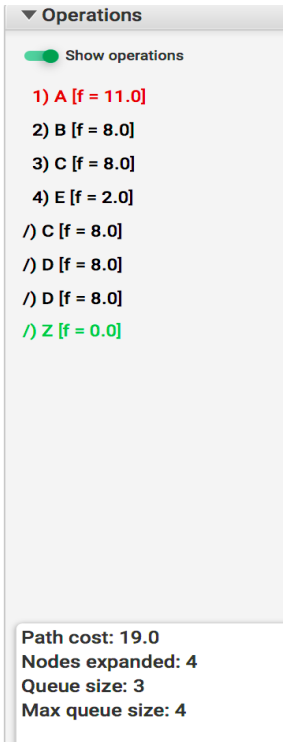
Esercizio 4



Esercizio 5

Con la strategia (Greedy) Best First i nodi espansi sono indicati in figura (ABCEZ), il costo della soluzione ACEZ è 19). La soluzione non è ottima (sarebbe ottima ACDZ costo 18) e questa condizione non è infatti garantita dalla strategia best-first.





Si noti che una soluzione alternativa potrebbe essere generata anche espandendo i nodi: ABCEZ dove il nodo C è, in questo caso, quello a sinistra generando quindi la soluzione ABCEZ con costo 35 (ovviamente anche questa non ottima).

Esercizio 6

Vedi slides del corso.