COMPITO DI FONDAMENTI DI INTELLIGENZA ARTIFICIALE
INTELLIGENZA ARTIFICIALE (v.o.) – PARTE I

25 Giugno 2008 (Tempo a disposizione 2h ; su 32 punti)

Esercizio 1 (punti 3)
Dire quali tra le seguenti formule nella logica del primo ordine sono una rappresentazione adeguata della frase:

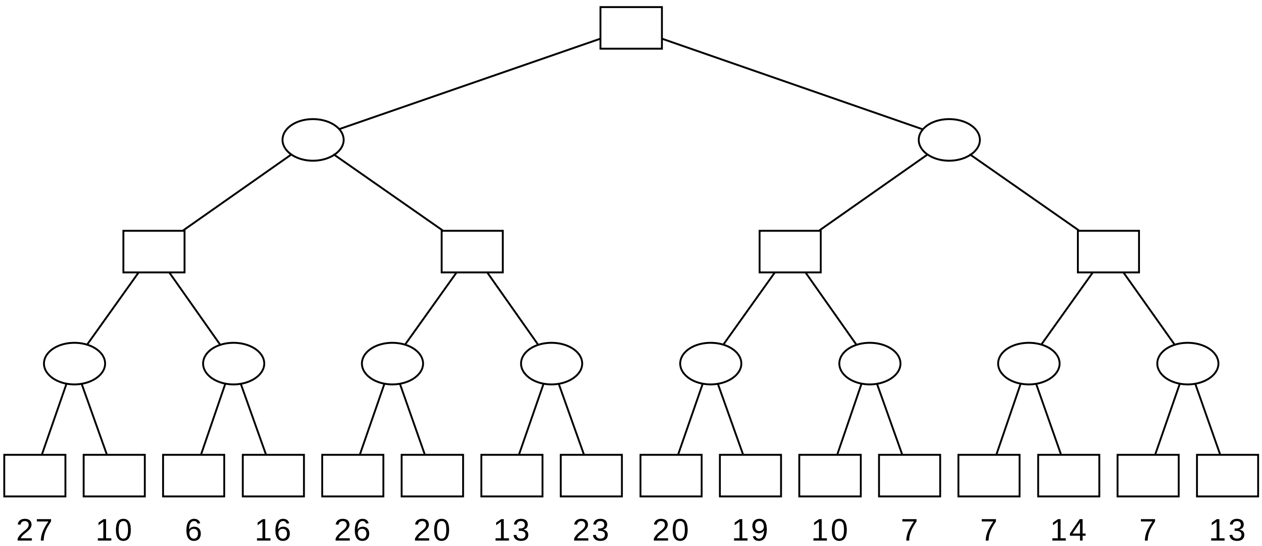
I bambini che fanno volare gli aquiloni sono felici.

Discutere e motivare le risposte.
b1. $\forall x\ \forall y\ [(bimbo(x) \wedge faVolare(x,y)) \rightarrow (aquilone(y) \wedge felice(x))]$
b2. $\forall x\ \forall y\ [(bimbo(x) \wedge aquilone(y) \wedge faVolare(x,y)) \rightarrow felice(y)]$
b3. $\forall x\ \exists y\ [(bimbo(x) \wedge aquilone(y) \wedge faVolare(x, y)) \rightarrow felice(x)]$

Esercizio 2 (punti 6)
Si consideri la seguente base di conoscenza Prolog:
```
insdiff(X,L):- not(not(L=[])),!,L=[X|_].
insdiff(X,[H|T]):- insdiff(X,T).
```
Si mostri l'albero di derivazione SLDNF relativo al goal `insdiff(3,[1,2|R])` e si dica qual è la risposta calcolata.

Esercizio 3 (punti 6)

Si consideri il seguente albero di gioco, in cui i punteggi sono dal punto di vista del primo giocatore (Max):



Si mostri come l'algoritmo min-max risolve il problema. Si mostrino poi i tagli alfa-beta.

Esercizio 4 (punti 3)
Supponiamo che per un problema di ricerca la cui funzione di costo è $f(n)$, siano disponibili due euristiche $h1(n)$ e $h2(n)$ entrambe ammissibili. Dire quali fra le seguenti euristiche $h(n)$ è ammissibile. Giustificare ogni risposta.
(a) $h(n) = max(h1(n); 2 * h2(n))$
(b) $h(n) = max(0.5 * h1(n); h2(n))$
(c) $h(n) = 0.5 * (h1(n) + h2(n))$

Esercizio 5 (punti 6)

Il problema della ripartizione: Il problema della ripartizione è definito nel seguente modo: è data in input una lista *L* di numeri interi positivi. L'output può essere o una ripartizione di *L* in due insiemi *U* e *V* tali che la somma degli elementi di *U* sia pari alla somma degli elementi di *V* (e quindi pari alla metà della somma degli elementi di *L*), o *FALSE* per indicare che tale ripartizione non esiste.
Per esempio, data la lista *L* = [1, 4, 6, 14, 17, 20] come input, l'output sarà *U* = [1, 4, 6, 20] e *V* = [14, 17] poiché entrambi gli insiemi sono tali che la somma dei loro elementi è pari a 31.
Invece, se ad esempio è data in input la lista *L* = [3, 4, 6, 14, 17, 20], l'output del problema sarà *FALSE* poiché non esiste nessuna ripartizione di *L* del tipo voluto.
Si consideri la seguente formalizzazione del problema come problema di ricerca:

- sia *M* la somma degli elementi della lista *L*;
- un generico stato è rappresentato da una tripla <*L*1;*U*1; *V*1>, dove *L*1 è una lista e *U*1 e *V*1 sono insiemi. *L*1, *U*1 e *V*1 rappresentano una ripartizione della lista *L* (cioè sono tali che la loro unione dia *L* e che ogni elemento di *L* appartenga ad uno e uno solo tra *L*1, *U*1 e *V*1);
- le operazioni possibili su <*L*1;*U*1; *V*1> sono definite nel seguente modo:
    - *opU*: eliminare l'elemento dalla testa di *L*1 e aggiungerlo a *U*1, applicabile se la somma degli elementi nell'insieme risultante non supera *M/*2;
    - *opV* : eliminare l'elemento dalla testa di *L*1 e aggiungerlo a *V*1, applicabile se la somma degli elementi nell'insieme risultante non supera *M/*2.

1. Definire lo stato iniziale e lo stato obiettivo per il problema in esame.
2. Dire quanto vale il *branching factor*.
3. Costruire l'albero di ricerca generato con una ricerca in profondità per il problema in cui *L* = [1*;* 2*;* 3*;* 4].
4. Supporre che la lista *L* abbia lunghezza 50. Dire quanto vale la profondità massima dell'albero
5. (indicare un limite sulla dimensione massima dell'albero).

Esercizio 6 (punti 5)

Si scriva un programma Prolog `compatta(L1,L2)` che data una lista di interi `L1` sostituisce `N` occorrenze successive di un elemento `E` con il risultato di `N*E` e produce la lista `L2`.
<u>Esempi:</u>

```
?-compatta([1,1,1,2,1,2],L).
yes L=[3,2,1,2]

?-compatta([2,3,2,2,4,2],L).
yes L=[2,3,4,4,2]
```

SUGGERIMENTO
Si scriva compatta nel seguente modo:

```
compatta([],[]).
compatta([H|T], [El|T2]):-  conta_e_consuma(H, T, N, T3), El is N*H,
                                              compatta(T3, T2).
```

E si definisca opportunamente conta_e_consuma:

```
conta_e_consuma(H, L, N, Lcons)
```

che consuma L finchè trova in testa H, restituendo in Lcons la lista rimanente e in N il numero di occorrenze di H trovate.

Esercizio 7 (punti 3)

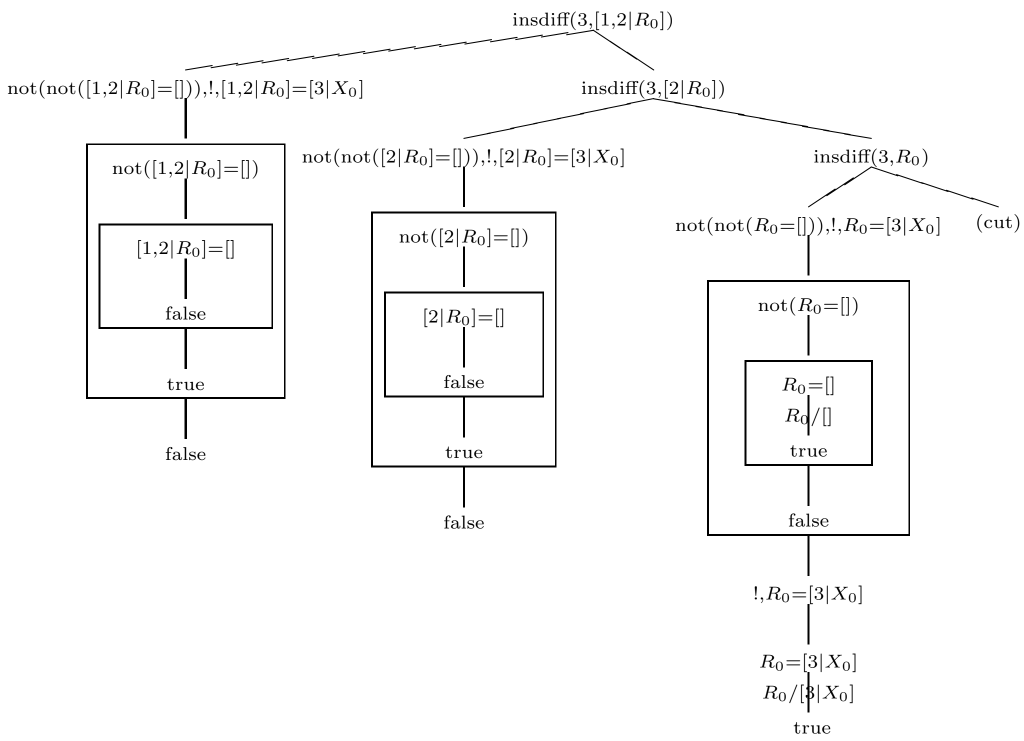
Si introducano e si commentino le strategie di ricerca lineare e lineare-input applicate alla risoluzione.

SOLUZIONE

Esercizio 1

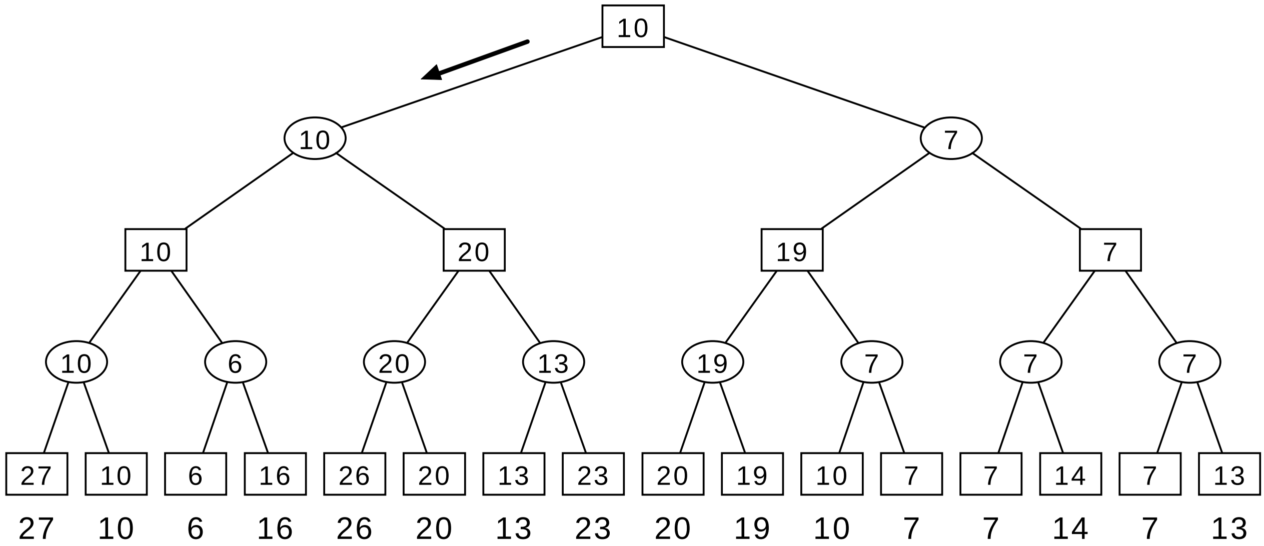
1. errata… vorrebbe dire che qualunque cosa un bambino faccia volare è un aquilone, e il bambino è felice.
2. errata... andrebbe tutto bene, ma purtroppo sono felici gli aquiloni e non i bimbi.
3. errata perché l'aquilone potrebbe non esistere, vista la posizione dell'esistenziale. Infatti, non è detto che per ciascun bambino esista un aquilone che il bambino fa volare.

Esercizio 2

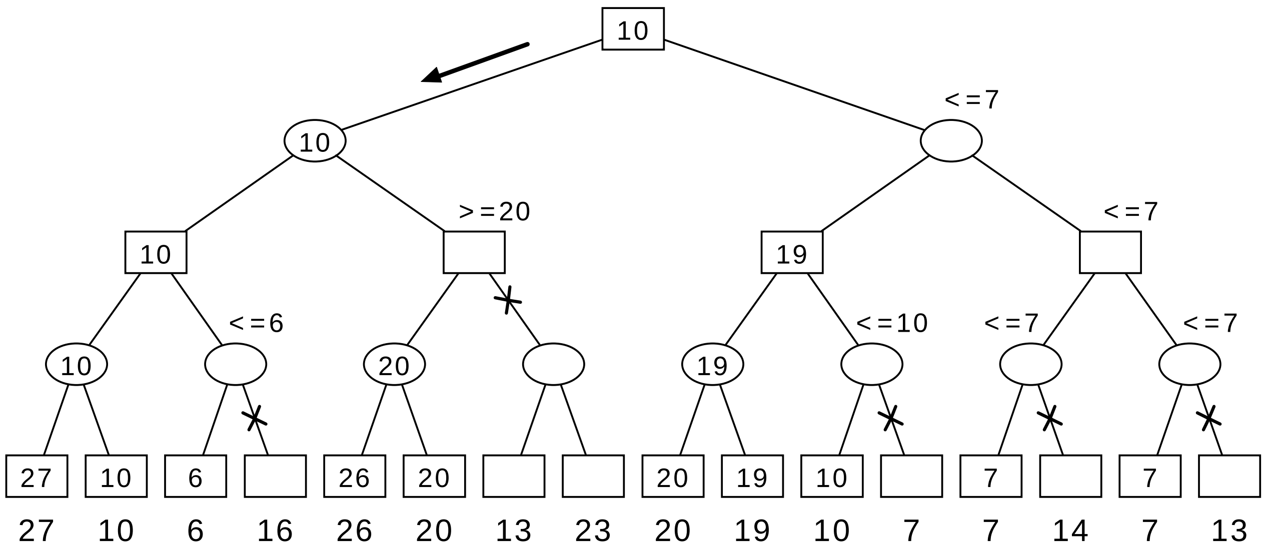


La risposta calcolata è $R/[3|X_0]$.

Esercizio 3
Min-max:

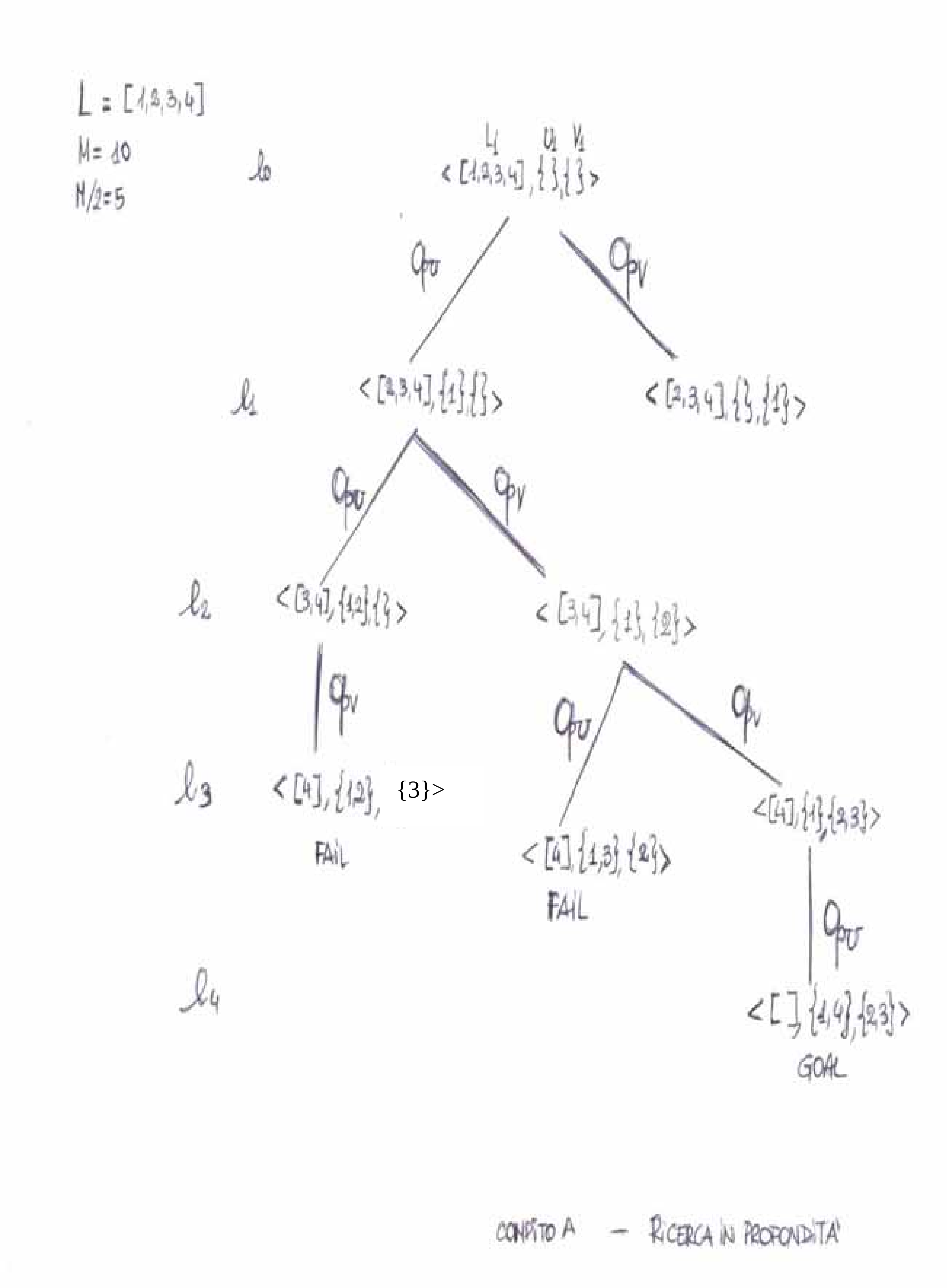


alfa-beta:



Esercizio 4
(a) $h(n) = max(h1(n); 2 * h2(n))$
non ammissibile, $2 * h2(n)$ potrebbe non essere una sottostima di f.
(b) $h(n) = max(0.5 * h1(n); h2(n))$
ammissibile, infatti $.5 * h1(n) < h1(n)$.
(c) $h(n) = 0.5 * (h1(n) + h2(n))$
ammissibile, è la media aritmetica delle due euristiche date che in generale per ogni nodo è minore della massima.

Esercizio 5
Lo stato iniziale è *<L; Ø; Ø>* Lo stato obiettivo è caratterizzato dall'avere *L*1 come lista vuota.
Il *branching factor* è 2 (2 operazioni possibili).
Albero di ricerca: vedi figura seguente.



Se la lista *L* ha lunghezza 50, la profondità massima è 50 (upper bound sulla dimensione massima dell'albero: 2^{50}).

Esercizio 6

```
%
% Data una lista L1, sostituisce N occorrenze successive di un elemento
% E con N*E e produce la lista L2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


compatta([],[]).
compatta([H|T], [El|T2]):-  conta_e_consuma(H, T, N, T3), El is N*H,
                                        compatta(T3, T2).
/* consuma L finchè trova in testa H, restituendo in Lcons la lista
rimanente e in N il numero di occorrenze di H trovate */

conta_e_consuma(H, L, N, Lcons):- conta_e_consuma(H, L, 1, N, Lcons).
conta_e_consuma(H, [H|T], Acc, N, Lcons):- !,

Acc2 is Acc + 1,

conta_e_consuma(H, T, Acc2, N, Lcons).
/* fine ricorsione: l'accumulatore è il valore totale, e la lista in
testa ha un elemento diverso da H, quindi ho finito di consumarla */

conta_e_consuma(H, Lcons, N, N, Lcons).



/* altra soluzione ancora: */

compatta(L1,L2):- compress(L1,L2).

compress([H|T], L2):- runcode([H|T], H, 0, L2).

runcode([], C, N, [N*C]).
runcode([H|T], H, N, Z) :-
        N1 is N+1,
runcode(T, H, N1, Z).
runcode([H|T], C, N, [N*C|Z]) :-
        H \== C,
runcode(T, H, 1, Z).


/* altra soluzione ricorsiva */
```

```
conta_e_consuma(H, [], 1, []).

conta_e_consuma(H, [H|T],  N, Lcons):-
!, conta_e_consuma(H, T,  N1, Lcons), N1  is Acc + 1.


conta_e_consuma(H, [_|T], N,1, T).
```