FONDAMENTI DI INTELLIGENZA ARTIFICIALE

25 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:

T*utti i bambini amano mangiare alcuni tipi di cibo e non amano mangiare altri tipi di cibo.*

OVVERO:

Per tutti i bambini esiste almeno un cibo X che mangiano e un cibo Y che NON mangiano.

Luca mangia ogni cibo.

Dimostrare con il metodo di risoluzione che: *Luca non è un bambino*.

Si usino i predicati:

bambino(X), X è un bambino

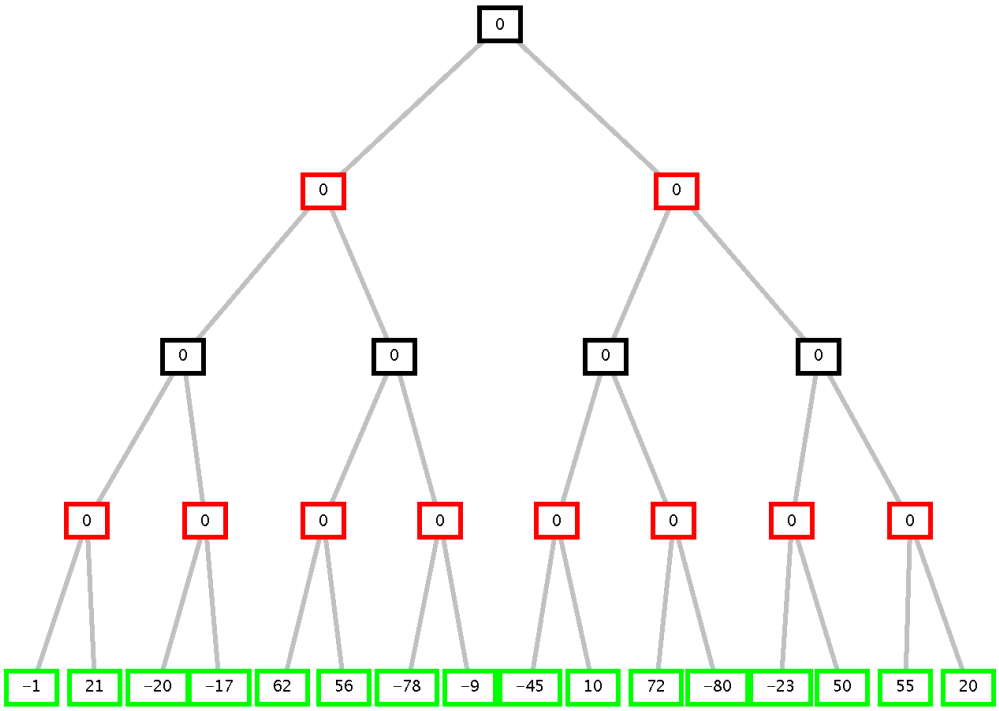
cibo(X), X è un cibo

mangia(X,Y), X mangia Y

e la costante: 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)

Data una lista di liste di interi **L1**, si scriva un programma Prolog **lastLists(L1,L2)** che restituisce in uscita una lista **L2** contenente tutti gli elementi in ultima posizione delle liste che compongono **L1**. Se una lista è vuota non verrà considerata.

A tale scopo si definisca anche il predicato Prolog **last(L,El)** che data una lista **L** non vuota ne restituisce l'ultimo elemento **El**.

Esempi:

```
?-lastLists([[1,2,7],[3],[],[4,5,6,9]],L2).
L2 = [7, 3, 9]
?-lastLists([],L2).
L2 = []
```

Esercizio 4 (5 punti)

Dato il seguente programma Prolog **select(L1,M,N, L2)** che data una lista di interi **L1** e due numeri interi **N** e **M**, con **N>=M**, crea una lista **L2**, inserendovi i numeri di **L1** maggiori di **M** e (*and*) minori di **N** (i numeri uguali a M e N non devono essere inseriti nella lista **L2)**.

select([],_,_, []).

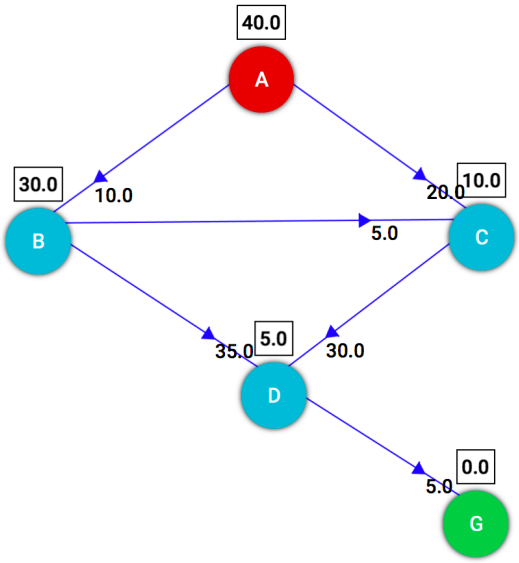
select([A|R],M, N, [A|Ra]) :- A<N,A>M, !, select(R,M,N, Ra).

select([_|R],M, N, Ra) :- select(R,M, N, Ra).

Si mostri l'*albero SLD* relativo al goal: **?-select([4,9],3,8,L)**. indicando i rami di fallimento, di successo e quelli tagliati dal cut. Si indichi anche la risposta calcolata per la variabile **L** 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;

Si indichi la condizione sulla stima euristica h(n) che garantisce l'ottimalità di A* su alberi e se è soddisfatta in questo caso.

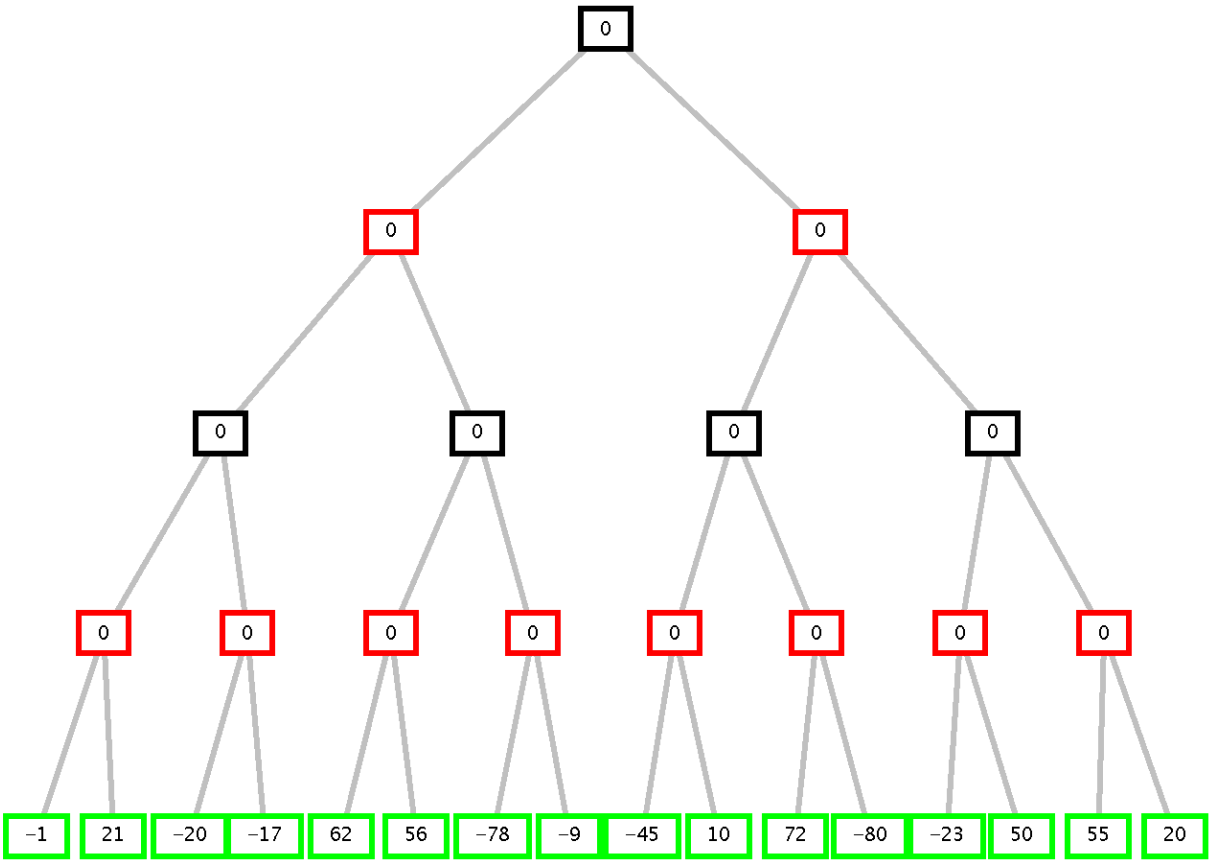
Esercizio 6 (5 punti)

Dopo avere brevemente introdotto l'algoritmo di Forward Checking, se ne mostri l'esecuzione su questo esempio:

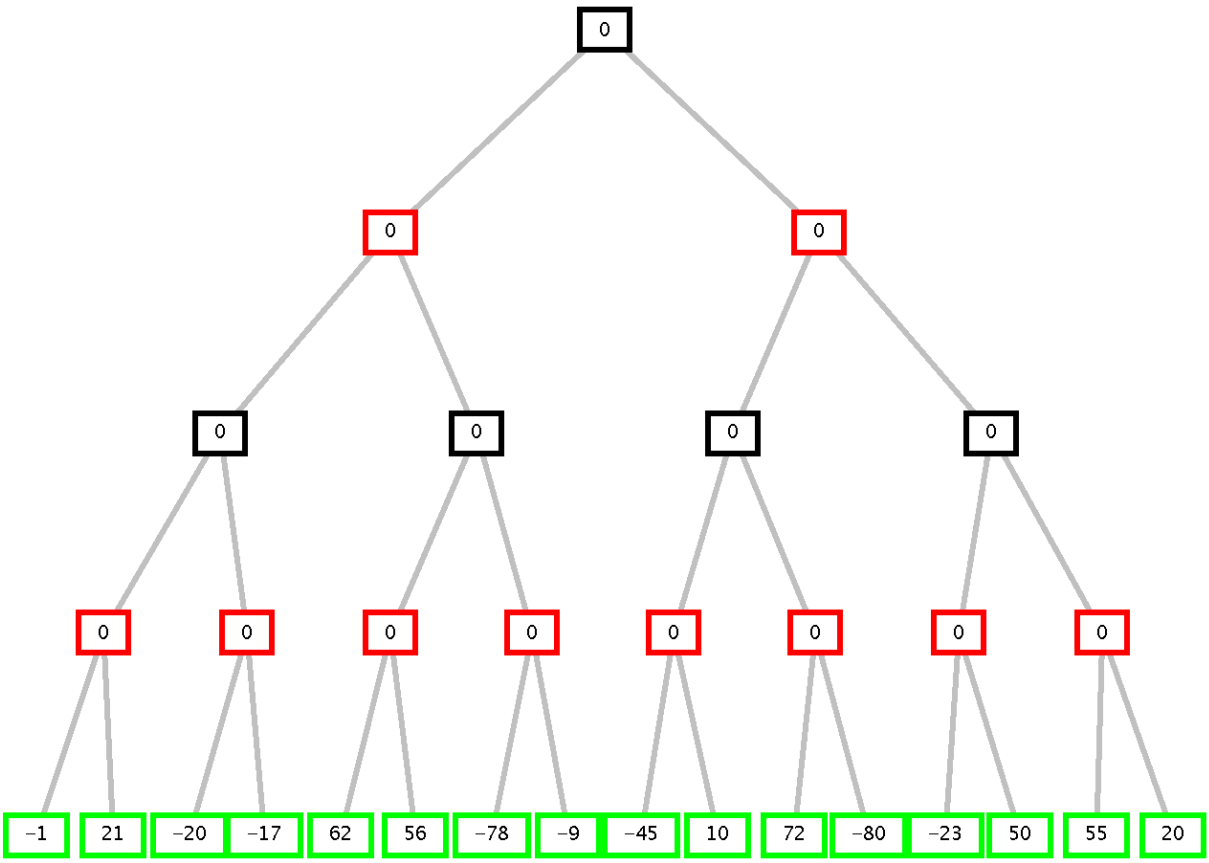
A::[1, 2, 3, 4, 5]
B::[1, 2, 3, 4, 5]
C::[1, 2, 3, 4, 5]
D::[1, 2, 3, 4, 5]
E::[1, 2, 3, 4, 5]
B>E
A<D
C<D
E<=B+A

considerando la prossima variabile da istanziare secondo l'euristica Minimum Remaining Values (MRV); a parità di MRV, si scelga la variabile che viene prima in ordine alfabetico. Nella scelta dei valori, si prediliga sempre di assegnare il valore minore. Si mostri la riduzione dei domini delle variabili future dopo ogni labeling, fino alla prima soluzione.

Min-max Cognome, Nome _____



Alfa-beta:



25 Gennaio 2024 - Soluzioni

Esercizio 1

Rappresentazione in FOL:

1. ∀Z bambino(Z) ⇒ ∃X (cibo(X) and mangia (Z, X)) ∧ ∃Y (cibo(Y) and not mangia(Z, Y))
2. ∀X cibo(X) ⇒ mangia(luca, X)

3. Query da dimostrare: not bambino(luca).

Trasformazione in forma a clausole:

Da formula: 1. ∀X bambino(X) ⇒ ∃Y (cibo(Y) and mangia (X, Y)) and ∃Z (cibo(Z) and not mangia(X, Z))
not bambino(X) or ((cibo(p1(X) and mangia(X, p1(X))) and ((cibo(p2(X) and mangia(X, p2(X)))

NOTA: p1(X) e p2(X) *funzioni skolem*

Distribuendo gli and rispetto all'or si ottengono quattro clausole:

C1a: not bambino(X) or cibo(p1(X))
C1b: not bambino(X) or mangia(X,p1(X))
C1c: not bambino(X) or cibo(p2(X))
C1d: not bambino(X) or not mangia(X,p2(X))
C2: not cibo(X) or mangia(luca, X)

Negando la query si ottiene:

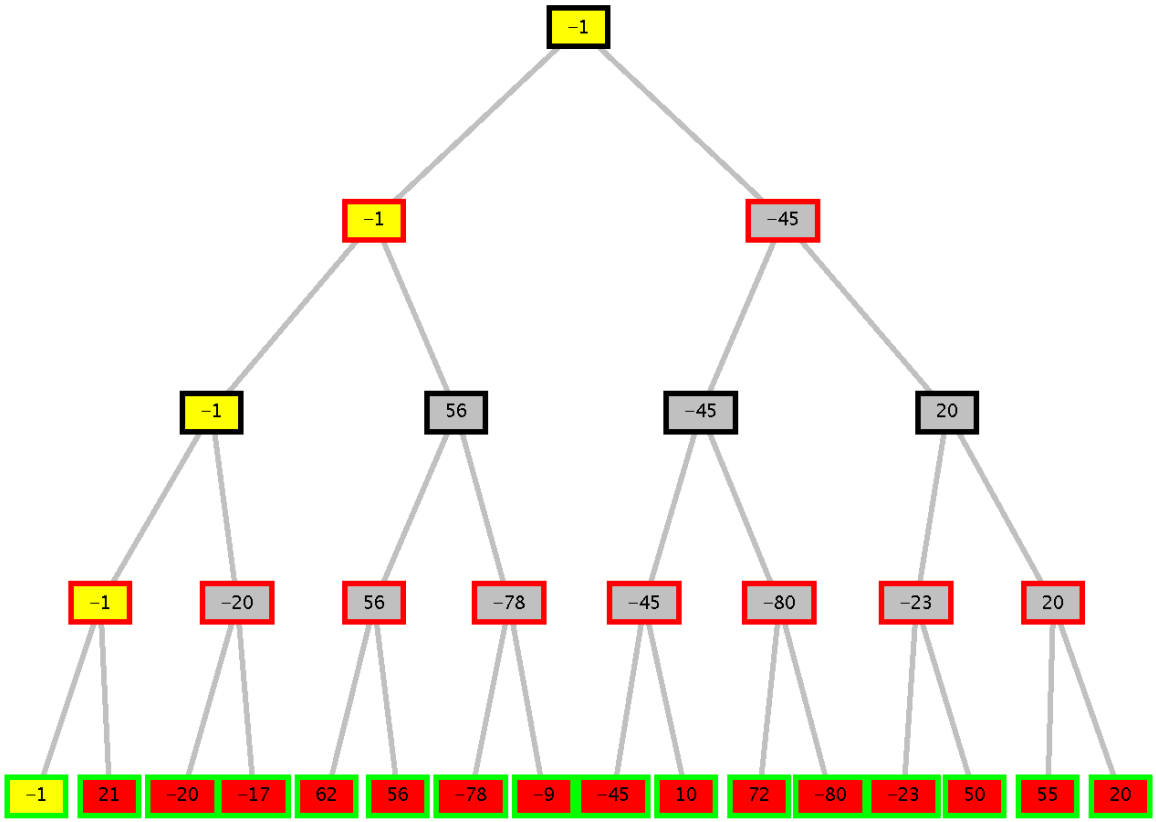
GNeg: bambino(luca).

Risoluzione:

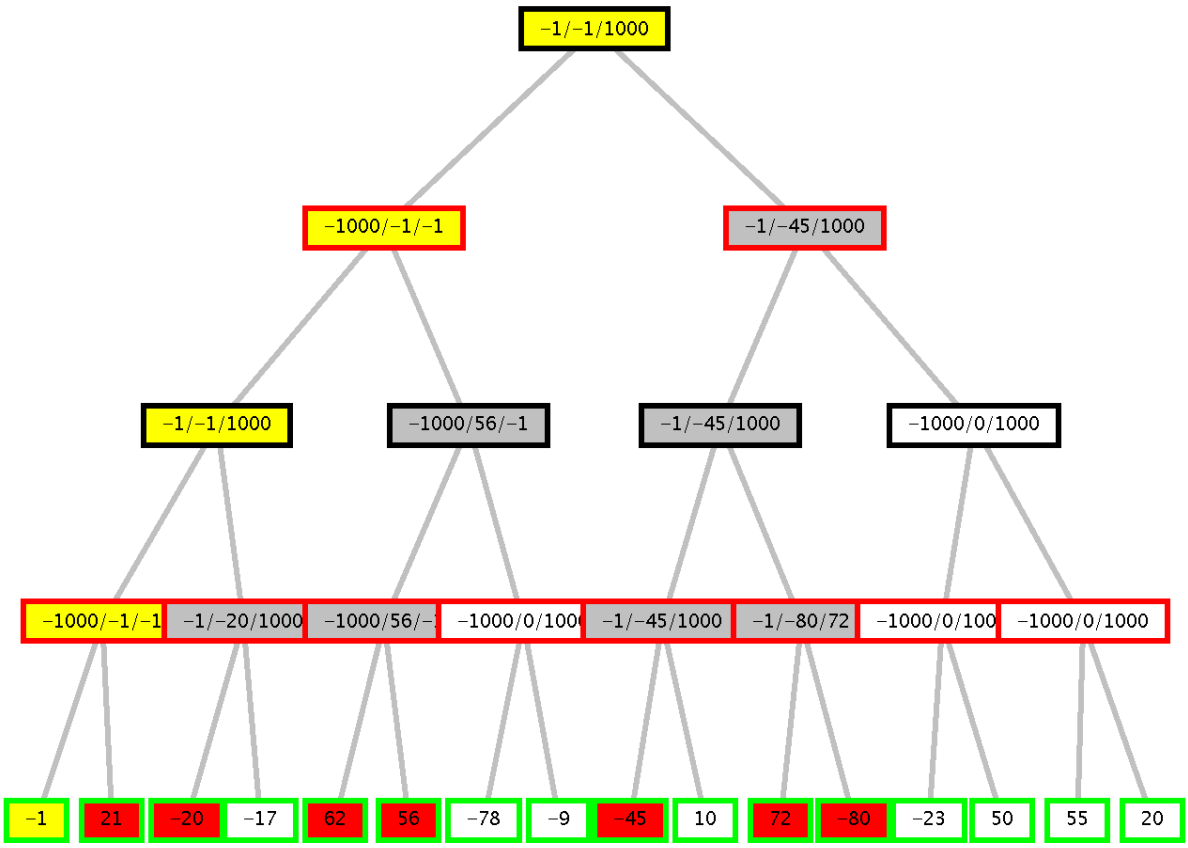
GNeg + C1d = C3: not mangia(luca,p2(luca))
C3 + C2 = C4: not cibo(p2(luca))
C4 + C1c = C5: not bambino(luca)
C5+ Gneg = contraddizione !!!

Esercizio 2

Min max:



Alfa Beta:



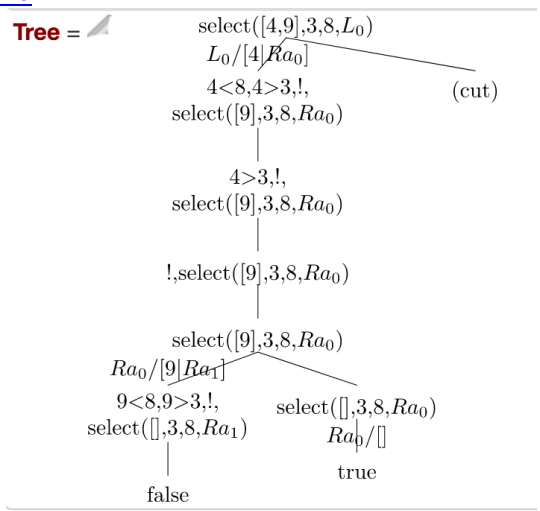
Esercizio 3

```
last([X], X) :- !.
last([_|Tail], Result) :- last(Tail, Result).

lastLists([],[]).
lastLists( [[]|T], T1) :- !, lastLists(T,T1).
lastLists( [X|T], [Z|T1]) :- last(X, Z), lastLists(T, T1).
```

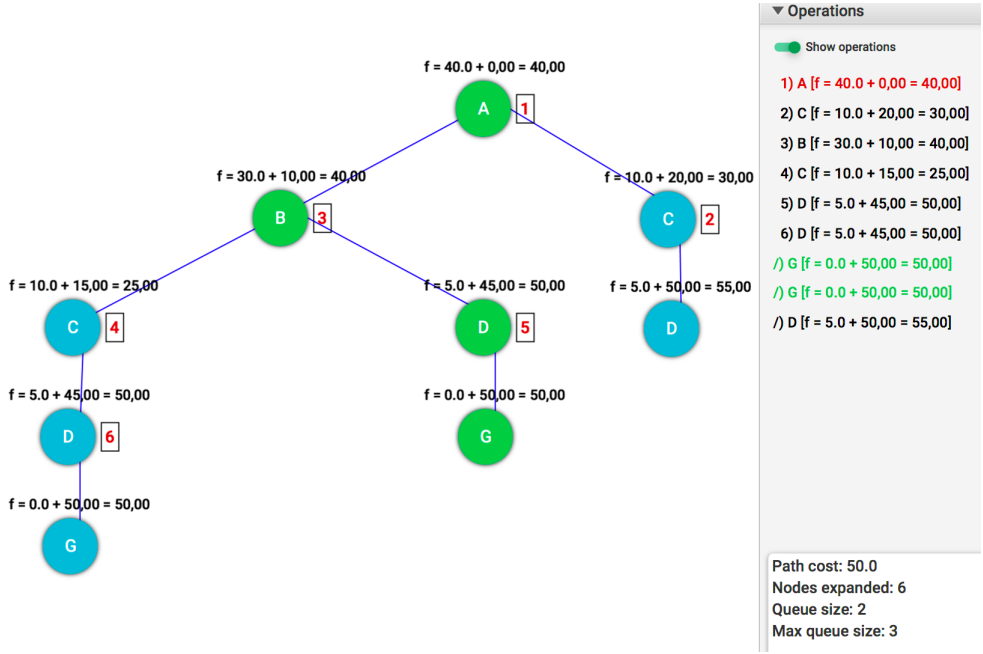
Esercizio 4

http://cplint.eu/p/mGWdAOCw.swinb



Esercizio 5

Con A*:



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

Vedi slide del corso per spiegare FC.

	A	**B**	**C**	**D**	**E**
Labeling	**A=1**	[1..5]	[1..5]	[1..5]	[1..5]
FC	A=1	[1..5]	[1..5]	**[2..5]**	[1..5]
Labeling	A=1	[1..5]	[1..5]	**D=2**	[1..5]
FC	A=1	[1..5]	**[1]**	D=2	[1..5]
Labeling	A=1	[1..5]	**C=1**	D=2	[1..5]
FC	A=1	[1..5]	C=1	D=2	[1..5]
Labeling	A=1	**B=1**	C=1	D=2	[1..5]
FC	A=1	B=1	C=1	D=2	**[ ] Fail**
Back e Labeling	A=1	**B=2**	C=1	D=2	[1..5]
FC	A=1	B=2	C=1	D=2	**[1]**
Labeling	A=1	B=2	C=1	D=2	**E=1**