Università degli Studi di Ferrara
Corso di Laurea in Ingegneria Elettronica

Algoritmo dei Tagli Alfa–Beta

Una implementazione in Java

Tesina per l'esame orale di
Intelligenza Artificiale
di
Tarin Gamberini

Corso di Intelligenza Artificiale (ante riforma 3+2)
Anno Accademico 2002/2003
Docente *E.Lamma*

Copyright (c) 2004 Tarin Gamberini.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, with the Front-Cover Texts being: "Università degli Studi di Ferrara", "Corso di Laurea in Ingegneria Elettronica", "Algoritmo dei Tagli Alfa–Beta", "Una implementazione in Java", "Tesina per l'esame orale di, Intelligenza Artificiale", "di", "Tarin Gamberini", "Corso di Intelligenza Artificiale (ante riforma 3+2)", "Anno Accademico 2002/2003", "Docente E.Lamma", and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Copyright (c) 2004 Tarin Gamberini.

È garantito il permesso di copiare, distribuire e/o modificare questo documento seguendo i termini della Licenza per Documentazione Libera GNU, Versione 1.1 o ogni versione successiva pubblicata dalla Free Software Foundation; senza Sezioni non Modificabili, con i Testi Copertina: "Università degli Studi di Ferrara", "Corso di Laurea in Ingegneria Elettronica", "Algoritmo dei Tagli Alfa–Beta", "Una implementazione in Java", "Tesina per l'esame orale di, Intelligenza Artificiale", "di", "Tarin Gamberini", "Corso di Intelligenza Artificiale (ante riforma 3+2)", "Anno Accademico 2002/2003", "Docente E.Lamma", e nessun Testo di Retro Copertina. Una copia della licenza è acclusa nella sezione intitolata "GNU Free Documentation License".

taringamberini [at] taringamberini [dot] com
www.taringamberini.com

Indice

Introduzione **7**

1 Principio di funzionamento **9**

2 Diagramma di flusso dell'algoritmo **13**

3 Analisi delle classi **17**
3.1 Descrittore dello stato di gioco 17
3.1.1 Attributi 17
3.1.2 Metodi 18
3.2 Tagli Alfa–Beta 18
3.2.1 Attributi 19
3.2.2 Metodi 19
3.3 Gioco generico 23
3.3.1 Attributi 24
3.3.2 Metodi 24

4 Verifica dell'implementazione in Java **27**
4.1 Esame del 25/06/2003 27
4.2 Esame del 18/03/2003 29
4.3 Esame del 18/03/2003 - variante 31

Conclusioni **33**

A Classe GameStatusDescriptor **35**

B Classe astratta TagliAlfaBeta **39**

C Classe GiocoGenericoConTagliAlfaBeta **45**
C.1 Esame del 25/06/2003 46
C.1.1 Metodo generaStatiFigli 46
C.1.2 Metodo valutaCostoMossa 51

C.2 Esame del 18/03/2003 52
C.2.1 Metodo generaStatiFigli 52
C.2.2 Metodo valutaCostoMossa 56

D Risultati delle verifiche **59**
D.1 Esame 25/06/2003 59
D.2 Esame del 18/03/2003 61
D.3 Esame del 18/03/2003 - variante 62

E Classi delle strutture dati **65**
E.1 Classe ListNode 65
E.2 Classe List 67
E.3 Classe GeneralTreeNode 72
E.4 Classe GeneralTree 74

F GNU Free Documentation License **79**

Elenco delle figure

1.1 Un figlio di C vale meno dei fratelli di C già esplorati. 10
1.2 Un figlio di C vale più dei fratelli di C già esplorati. 10

2.1 Diagramma di flusso dell'algoritmo dei Tagli Alfa–Beta 14

4.1 Spazio degli stati di gioco 27
4.2 Spazio degli stati ridotto con i Tagli Alfa–Beta 28
4.3 Spazio degli stati di gioco 29
4.4 Spazio degli stati ridotto con i Tagli Alfa–Beta 30
4.5 Spazio degli stati ridotto con i Tagli Alfa–Beta 32

Introduzione

Durante il corso di Intelligenza Artificiale abbiamo studiato una particolare categoria di giochi aventi le seguenti caratteristiche:

- Sono presenti solo due giocatori che eseguono alternativamente una mossa alla volta.
- Sono giochi a conoscenza perfetta: entrambi i giocatori hanno le stesse informazioni.

Tipici giochi a conoscenza perfetta sono la dama, gli scacchi, ecc..., mentre tipici giochi a conoscenza imperfetta sono quelli di carte come poker, bridge, ecc...

Lo svolgersi del gioco si può interpretare attraverso un albero, detto *spazio degli stati*, in cui ogni nodo rappresenta uno stato di gioco. Si comincia da uno stato iniziale in cui si è stabilito il giocatore che ha il diritto di compiere una mossa (tale diritto di mossa è detto in gergo *avere il gioco in mano* o più brevemente *avere la mano* o *essere di mano*). Il giocatore che è di mano attraverso una mossa raggiunge uno stato successivo in cui la mano passa all'altro giocatore. Così via fino al raggiungimento di uno stato finale: una foglia dell'albero dello spazio degli stati in cui non sia possibile alcuna altra mossa.

Un tipico approccio per risolvere un gioco consiste in una ricerca nello spazio degli stati condotta da algoritmi. Durante il corso abbiamo studiato gli algoritmi del MIN–MAX e quello dei Tagli Alfa–Beta.

In questa relazione abbiamo implementato l'algoritmo dei Tagli Alfa–Beta in linguaggio Java. Consci dell'esistenza di linguaggi in cui l'esecuzione sia più efficiente (C++) oppure l'implementazione sia più immediata (PROLOG), abbiamo scelto Java semplicemente per approfondirne la conoscenza.

La verifica della correttezza dell'implementazione è stata condotta su alcuni esercizi proposti in alcune sessioni d'esame e corredati di soluzione.

Capitolo 1

Principio di funzionamento

Il principio di funzionamento dell'algoritmo dei Tagli Alfa–Beta si basa sull'ipotesi che entrambi i giocatori siano *razionali* ed in quanto tali scelgano sempre la mossa migliore.

Il giocatore che è di mano sceglie la mossa ritenuta migliore in base ad un punteggio associatole da un'opportuna funzione di valutazione. La mossa migliore è quella associata al massimo fra i punteggi di tutte le mosse possibili.

Adottiamo la convenzione di indicare i due giocatori con i nomi simbolici MAX e MIN e di calcolare la mossa migliore attraverso un massimo per MAX, ed un minimo per MIN, fra i punteggi di tutte le possibili mosse.

Ricordiamo brevemente che l'algoritmo MIN–MAX, dato uno stato di gioco S, sceglie la mossa migliore esplorando *tutto* lo spazio degli stati generato a partire da S. Se b è il fattore di ramificazione dell'albero dello spazio degli stati e d è il livello massimo di profondità (la radice ha livello di profondità 0) allora il numero di nodi esplorati è b^d.

L'algoritmo dei Tagli Alfa–Beta *tenta* di ridurre il numero di nodi esplorati attraverso una semplice osservazione basata sulla *razionalità* dell'avversario.

Consideriamo la situazione riportata in figura 1.1. Nello stato A è di mano il giocatore MAX. Esplorato lo stato B con punteggio 10, MAX procede con l'esplorazione del nodo C. Nel caso in cui MAX scopra che C valga un punteggio < 10, la mossa migliore che potrebbe fare è quella di raggiungere B.

MAX esplora C e poi D con valore 5. MAX, sapendo che MIN agirà in modo *razionale*, attribuisce a C un punteggio ≤ 5. Infatti MIN raggiungerebbe D solo se dopo aver esplorato gli altri stati figli di C essi avessero tutti un punteggio ≥ 5; nel caso in cui tra i figli di C esistesse uno stato X con punteggio $x < 5$, MIN raggiungerebbe X.

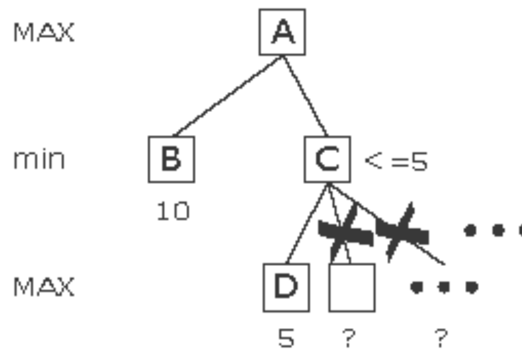

Figura 1.1: Un figlio di C vale meno dei fratelli di C già esplorati.

Pertanto MAX *appena scopre* che C vale meno del suo fratello B *taglia* dall'albero tutti i figli di C *evitando* di esplorarli.

L'efficienza dell'algoritmo dei Tagli Alfa-Beta dipende da *quando* un giocatore scopre di poter tagliare stati dall'albero. Se come in figura 1.2 D vale

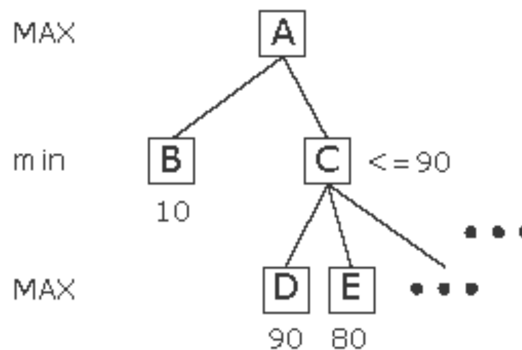

Figura 1.2: Un figlio di C vale più dei fratelli di C già esplorati.

90, MAX scopre che C vale un punteggio ≤ 90. Tuttavia MAX ora non può affermare che raggiungere B sia più vantaggioso. MAX deve necessariamente esplorare gli altri figli di C. Quando, e solo quando, troverà fra di essi uno stato V di valore $v < 10$ potrà tagliare quelli successivi, poichè avrà appena scoperto che raggiungere B è più vantaggioso.

Nell'ipotesi peggiore in cui punteggi migliori siano trovati per ultimi, l'efficienza dei Tagli Alfa–Beta è la stessa di quella dell'algoritmo MAX–MIN, ossia b^d. Nell'ipotesi migliore in cui i punteggi migliori siano trovati per primi, l'efficienza è $b^{d/2}$. Riducendo della radice quadrata il fattore di ramificazione si può guardare due volte più avanti nello stesso tempo. Mediamente, assumendo una distribuzione casuale dei punteggi degli stati, l'efficienza è circa $b^{3d/4}$.

Si noti che tali risultati valgono se l'albero è "ideale": con numero di ramificazioni e profondità fissate per ogni nodo.

Il principio di funzionamento dell'algoritmo dei Tagli Alfa–Beta è stato spiegato in riferimento alla figure 1.1 e 1.2 in cui MAX è il giocatore inizialmente di mano. Si possono trovare situazioni in cui sia MIN inizialmente di mano e che possono essere spiegate in maniera analoga: sarà MIN a tener conto dell'agire *razionale* di MAX.

Capitolo 2

Diagramma di flusso dell'algoritmo

Una descrizione indicativa dell'algoritmo dei Tagli Alfa–Beta è data nel diagramma di flusso in figura 2.1.

Si tratta di un algoritmo ricorsivo che esplora l'albero dello spazio degli stati di gioco con strategia depth first, potando se necessario.

1. Il nodo esplorato viene etichettato con il valore simbolico $+\infty$ se nello stato corrente è di mano MIN, o $-\infty$ se è di mano MAX.

2. Si generano gli eventuali stati figli. Analizzando lo stato corrente si scoprono quali mosse sono applicabili individuando così gli stati figli raggiungibili.

3. Se esistono stati figli e non è stata ancora raggiunta la massima profondità di esplorazione dell'albero si prosegue, altrimenti si salta al punto 9.

4. Se esistono ancora figli da esaminare si prosegue, altrimenti si salta al punto 8.

5. Si esplora ricorsivamente lo stato figlio, ricominciando dal punto 1.

6. Giunti ad uno stato foglia si chiude una ricorsione e si ritorna in quella dello stato padre della foglia. A tale stato occorre associare un punteggio provvisorio calcolato come massimo, se è di mano MAX, o come minimo, se è di mano MIN, fra il valore della foglia e quello provvisorio del padre.

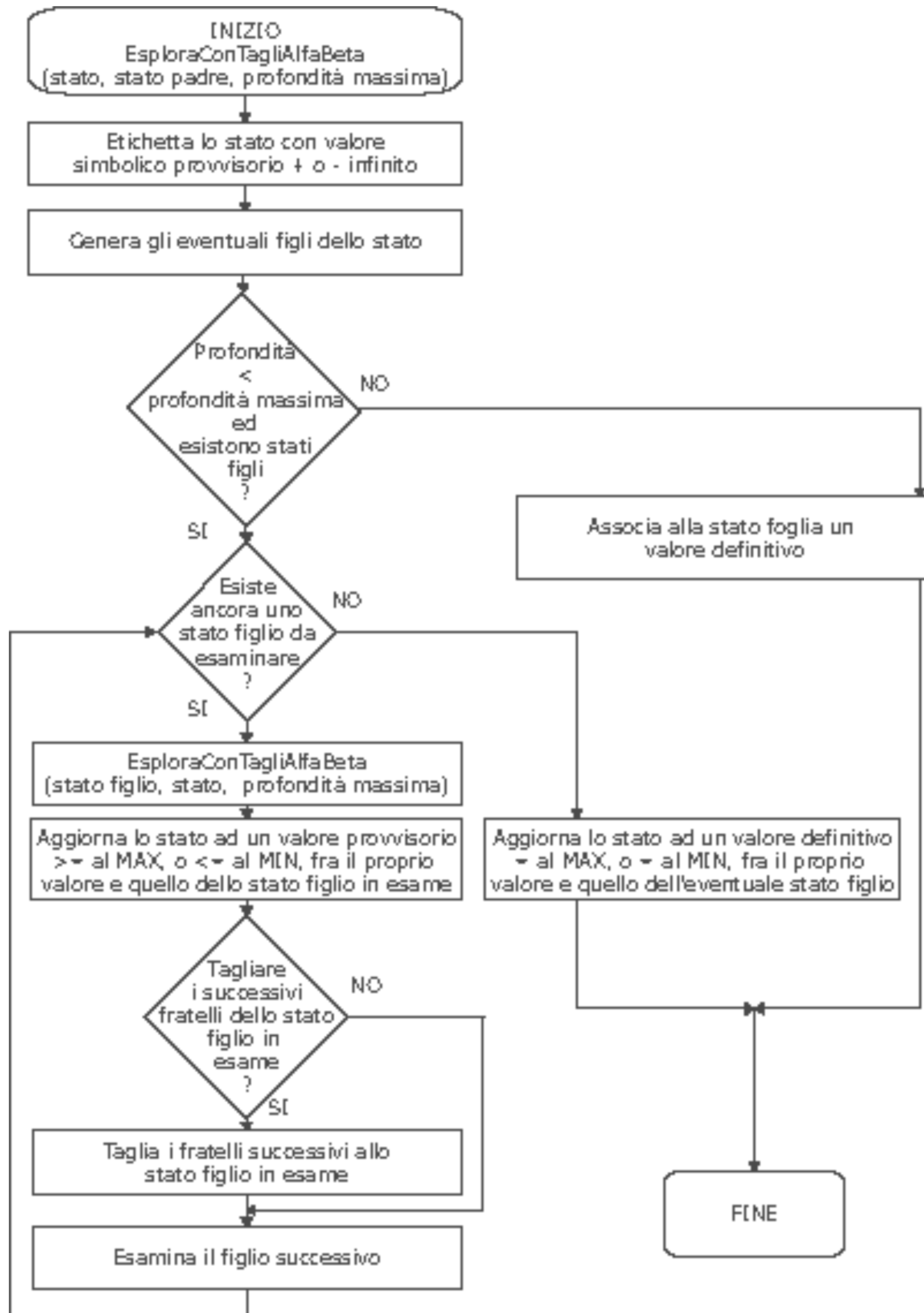

Figura 2.1: Diagramma di flusso dell'algoritmo dei Tagli Alfa–Beta

7. Se esistono le condizioni per tagliare i fratelli della foglia non ancora esplorati si taglia l'albero, altrimenti si prosegue esaminando il figlio successivo.

8. Se non esistono altri figli da esaminare occorre associare allo stato padre un punteggio definitivo calcolato come massimo, se è di mano MAX, o come minimo, se è di mano MIN, fra il valore della foglia e quello provvisorio del padre.

9. Se invece lo stato non ha figli, ovvero è uno foglia, oppure è stata raggiunta la massima profondità di esplorazione fissata, si assegna allo stato un valore definitivo.

Capitolo 3

Analisi delle classi

3.1 Descrittore dello stato di gioco

Ogni stato di gioco è rappresentato da un descrittore appartenente alla classe `GameStatusDescriptor`, il cui codice sorgente è consultabile dettagliatamente in appendice A.

3.1.1 Attributi

Lo stato di gioco è descritto attraverso i seguenti attributi:

private Object gameStatus È il riferimento allo stato del gioco. Nell'ottica di rendere la classe *riusabile* `gameStatus` si riferisce ad un oggetto generico poiché ogni gioco può richiedere una propria specifica descrizione dello stato. Per semplicità nelle nostre simulazioni si è descritto lo stato con un carattere, ovvero un oggetto della classe Character;

private Float value Valore del nodo associato allo stato `gameStatus`;

private String moveDone Descrizione della mossa compiuta per raggiungere lo stato `gameStatus`;

private String hasToMove Descrizione del giocatore che nello stato `gameStatus` è di mano e che deve quindi compiere una mossa. Abbiamo scelto i nomi simbolici MAX e MIN.

private String label Un'etichetta simbolica con i seguenti significati:

- NON_MARCATO Il nodo `gameStatus` non è ancora stato esplorato ed i valori degli attributi di un oggetto della classe `GameStatusDescriptor` non sono significativi.

- ASSEGNATO_VALORE Il nodo `gameStatus` è stato completamente esplorato, compresi tutti gli eventuali suoi sotto–stati. Tutti i valori degli attributi di un oggetto della classe `GameStatusDescriptor` sono significativi, in particolare il campo `value` rappresenta il punteggio *definitivo* associato al nodo.
- PIU_INFINITO Il nodo `gameStatus`, in cui è di mano MIN, è in fase di esplorazione. I valori degli attributi sono significativi ad eccezione di `value`.
- MENO_INFINITO Il nodo `gameStatus`, in cui è di mano MAX, è in fase di esplorazione. I valori degli attributi sono significativi ad eccezione di `value`.
- MAGGIORE_O_UGUALE Il nodo `gameStatus`, in cui è di mano MAX, è in fase di esplorazione. L'attributo `value` è un punteggio *provvisorio* ottenuto in seguito alla completa esplorazione di almeno un figlio di `gameStatus`.
- MINORE_O_UGUALE Il nodo `gameStatus`, in cui è di mano MIN, è in fase di esplorazione. L'attributo `value` è un punteggio *provvisorio* ottenuto in seguito alla completa esplorazione di almeno un figlio di `gameStatus`.

3.1.2 Metodi

I comportamenti che la classe `gameStatusDescriptor` mette a disposizione sono quelli "classici": creazione, set e get degli attributi. Si rimanda all'appendice A per maggiori dettagli.

3.2 Implementazione in linguaggio Java dei Tagli Alfa–Beta come classe astratta

Nell'ottica di rendere la classe *riusabile* per vari tipologie di giochi è stata implementata come classe astratta, il cui codice sorgente è consultabile dettagliatamente in appendice B.

Ogni gioco ha i propri specifici insiemi di mosse e di stati. Per questo si delega all'utente il compito di ridefinire il metodo `generaDescrittoriStatiFigli`:

```
public abstract
List generaDescrittoriStatiFigli( GameStatusDescriptor currentGSD );
```

che ricevuto un descrittore `currentGSD` dello stato corrente di gioco, lo *analizza* per scoprire le eventuali mosse possibili, e ritorna una lista di stati figli ognuno dei quali è raggiungibile applicando la relativa mossa.

Ogni gioco ha un suo modo di attribuire punteggi alle mosse; di più, per uno stesso gioco possono essere progettati sistemi di valutazione del costo delle mosse più o meno raffinati. Per questo si delega all'utente il compito di ridefinire anche il metodo `valutaCostoMossa`:

```
public abstract
Float valutaCostoMossa( String moveToDone );
```

che ricevuto una stringa rappresentativa della mossa da compiere ne ritorna il costo.

3.2.1 Attributi

La classe rende disponibile gli attributi di istanza:

```
private GeneralTree p;
private int profonditaMassima;
```

La prima è un riferimento all'albero dello spazio degli stati di gioco, la seconda è la profondità massima di esplorazione a cui si intende scendere nell'esplorazione dell'albero.

3.2.2 Metodi

TagliAlfaBeta

Il costruttore della classe riceve in ingresso la profondità massima a cui si intende scendere nell'esplorazione dell'albero e lo stato iniziale di gioco:

```
public TagliAlfaBeta
( int profonditaMassimaFissata, GameStatusDescriptor gsdStatoIniziale )

p = new GeneralTree( "albero dello spazio degli stati", gsdStatoIniziale );

esploraConTagliAlfaBeta( p , null , 0 );
```

Quindi crea un albero, inizialmente vuoto, per memorizzare lo spazio degli stati e successivamente invoca il metodo `esploraConTagliAlfaBeta` passando: il riferimento `p` all'albero, nessun riferimento al padre di `p` (la radice non ha padre) e la profondità corrente di esplorazione (la radice ha profondità nulla).

esploraConTagliAlfaBeta

Il metodo è ricorsivo ed effettua una visita depth first con eventuale potatura.

```
private void esploraConTagliAlfaBeta
( GeneralTree p, GeneralTree padreDiP, int profonditaCorrente )
```

`p` è il riferimento all'albero corrente che si desidera esplorare, `padreDiP` è il riferimento al padre di `p`, infine `profonditaCorrente` indica la profondità corrente di esplorazione.

Lo stato corrente viene etichettato con valore simbolico $\pm\infty$, quindi si generano gli eventuali stati figli:

```
etichettaConValoreSimbolico( p );

List DescrittoriStatiFigliPossibili =
generaDescrittoriStatiFigli(
  ( GameStatusDescriptor ) p.getRoot().getObject() );
```

`generaDescrittoriStatiFigli` si occupa anche di assegnare un punteggio allo stato creato, vedremo più avanti come.

Ora se non è stata raggiunta la massima profondità di esplorazione ed esistono stati figli si prosegue:

```
if ( profonditaCorrente < profonditaMassima &&
     !DescrittoriStatiFigliPossibili.isEmpty() ) {
     ...
     }
else
     ( ( GameStatusDescriptor ) p.getRoot().getObject() )
       .setLabel( GameStatusDescriptor.ASSEGNATO_VALORE );
```

Altrimenti è impossibile esplorare ulteriormente l'albero ed il valore provvisorio assegnato allo stato viene reso definitivo applicando l'etichetta `ASSEGNATO_VALORE`.

Si prosegue esplorando gli stati figli, ma prima occorre collegarli allo stato corrente `p`, ed occorre inizializzare il puntatore `pDescrittoreFiglioCorrente` al primo elemento della lista dei figli. Per facilitare il taglio di eventuali figli il contatore `posizioneDescrittoreFiglioCorrente` memorizza la posizione dello stato corrente all'interno della lista:

```
p.appendListOfSons( p, DescrittoriStatiFigliPossibili );
ListNode pDescrittoreFiglioCorrente = p.getRoot().getNexts().getFirst();
int posizioneDescrittoreFiglioCorrente = 1;
```

Mentre ci sono figli si scorre la lista. Per ogni figlio lo si esplora invocando ricorsivamente il metodo `esploraConTagliAlfaBeta` passando nell'ordine: il figlio da esplorare, che diventerà il futuro stato corrente, lo stato corrente, che diventerà il futuro stato padre, e la profondità di esplorazione incrementata di una unità:

```
while ( pDescrittoreFiglioCorrente != null ) {
  esploraConTagliAlfaBeta(
    ( GeneralTree ) pDescrittoreFiglioCorrente.getObject(),
    p, profonditaCorrente + 1 );
```

Raggiunto uno stato a profondità massima o uno stato foglia le ricorsioni terminano, il ciclo `while` prosegue aggiornando il valore dello stato corrente `p` confrontandolo con quello del figlio. Se è il caso di tagliare i figli di `p` ancora inesplorati si invoca il metodo `cutFromPosition` a cui si passa la posizione successiva a quella del figlio appena esplorato:

```
  aggiornaValore( p,
    ( GeneralTree ) pDescrittoreFiglioCorrente.getObject() );

  if ( valutaSeTagliare( p, padreDiP ) ) {
    System.out.println(
      "Tagliato il ramo che porta allo stato:" +
      ( ( GeneralTree ) DescrittoreFiglioCorrente.getNext().getObject() )
      .getRoot().getObject().toString() );
    p.getRoot().getNexts().cutFromPosition(
      posizioneDescrittoreFiglioCorrente + 1 );
}
pDescrittoreFiglioCorrente = pDescrittoreFiglioCorrente.getNext();
posizioneDescrittoreFiglioCorrente++;
```

Altrimenti si procede col figlio successivo. Da notare che in caso di taglio il metodo `cutFromPosition` rende il figlio correntemente esplorato l'ultimo della lista dei figli, per cui passando al figlio successivo (null) il ciclo `while` termina. In questo modo *non vengono esplorati i nodi tagliati* via dallo spazio degli stati.

etichettaConValoreSimbolico

I metodo ricevuto uno stato lo etichetta con valore simbolico $+\infty$ se è di mano MIN, $-\infty$ se è di mano MAX:

```
private void etichettaConValoreSimbolico(GeneralTree p) {

if ( ( ( GameStatusDescriptor ) p.getRoot().getObject() )
  .getHasToMove().equals( GameStatusDescriptor.MAX ) )

  ( ( GameStatusDescriptor ) p.getRoot().getObject() )
    .setLabel( GameStatusDescriptor.MENO_INFINITO );

else
  ( ( GameStatusDescriptor ) p.getRoot().getObject() )
    .setLabel( GameStatusDescriptor.PIU_INFINITO );
}
```

aggiornaValore

Riceve in ingresso uno stato padre `pFather` ed uno figlio `pSon`. Aggiorna il valore di `pFather` al massimo, se in `pFather` è di mano MAX, o al minimo, se in `pFather` è di mano MIN, fra il valore del padre e quello figlio.

L'aggiornamento di un nodo passa attraverso varie fasi, se è di mano MAX:

1. Il padre con valore provvisorio $-\infty$ viene marcato con valore provvisorio $\geq$ al valore del figlio.

2. Il padre con valore provvisorio $\geq$ viene marcato con valore provvisorio $\geq$ al massimo fra il valore del padre e quello del figlio.

```
private void aggiornaValore(GeneralTree pFather, GeneralTree pSon) {

GameStatusDescriptor pFatherGSD =
  ( GameStatusDescriptor ) pFather.getRoot().getObject();
GameStatusDescriptor pSonGSD =
  ( GameStatusDescriptor ) pSon.getRoot().getObject();

if ( pFatherGSD.getHasToMove().equals( GameStatusDescriptor.MAX ) )
  if ( pFatherGSD.getLabel().equals( GameStatusDescriptor.MENO_INFINITO ) ) {
    pFatherGSD.setLabel( GameStatusDescriptor.MAGGIORE_O_UGUALE );
    pFatherGSD.setValue( pSonGSD.getValue() );
  }
  else
    pFatherGSD.setValue( massimo( pFatherGSD.getValue(), pSonGSD.getValue() ) );

else
  if ( pFatherGSD.getLabel().equals( GameStatusDescriptor.PIU_INFINITO ) ) {
    pFatherGSD.setLabel( GameStatusDescriptor.MINORE_O_UGUALE );
    pFatherGSD.setValue( pSonGSD.getValue() );
  }
  else
    pFatherGSD.setValue( minimo( pFatherGSD.getValue(), pSonGSD.getValue() ) );
}
```

Analogamente se è di mano MIN:

1. Il padre con valore provvisorio $+\infty$ viene marcato con valore provvisorio $\leq$ al valore del figlio.

2. Il padre con valore provvisorio $\leq$ viene marcato con valore provvisorio $\leq$ al minimo fra il valore del padre e quello del figlio.

valutaSeTagliare

Riceve in ingresso lo stato corrente `p` e lo stato padre `pFather`:

```
private boolean valutaSeTagliare( GeneralTree p, GeneralTree pFather )
```

Ritorna `true` se è il caso di tagliare i figli di `p` ancora inesplorati.

Se lo stato figlio non è la radice si considerano i descrittori dello stato del padre e del figlio:

```
    if ( pFather != null ) {
      GameStatusDescriptor pGSD =
        ( GameStatusDescriptor ) p.getRoot().getObject();
      GameStatusDescriptor pFatherGSD =
        ( GameStatusDescriptor ) pFather.getRoot().getObject();
```

se i valori provvisori fra padre e figlio individuano insiemi ad intersezione nulla è il caso di tagliare. Ciò si verifica se il padre vale $\geq a$ ed il figlio vale $\leq b$ con $a \geq b$, oppure se il padre vale $\leq a$ ed il figlio vale $\geq b$ con $a \leq b$:

```
  if ( pFatherGSD.getLabel().equals( GameStatusDescriptor.MAGGIORE_O_UGUALE )
       &&
       pGSD.getLabel().equals( GameStatusDescriptor.MINORE_O_UGUALE ) ) {
    if ( pFatherGSD.getValue().compareTo( pGSD.getValue() ) >= 0 )
      return true;
  }
  else
  if ( pFatherGSD.getLabel().equals( GameStatusDescriptor.MINORE_O_UGUALE )
       &&
       pGSD.getLabel().equals( GameStatusDescriptor.MAGGIORE_O_UGUALE ) ) {
    if ( pFatherGSD.getValue().compareTo( pGSD.getValue() ) <= 0 )
      return true;
  }
```

Occorre ritornare `false` sia nel caso in cui i valori provvisori fra padre e figlio non individuino insiemi ad intersezione nulla, sia nel caso in cui il figlio sia la radice, che ovviamente non ha fratelli da tagliare:

```
    return false;
    }
      /* Se il padre è null allora stiamo analizzando la radice */
      return false;
  }
```

getAlberoRisultato

Ritorna un riferimento all'albero. Si noti che il costruttore `TagliAlfaBeta` invoca immediatamente il metodo `esploraConTagliAlfaBeta` quindi `getAlberoRisultato` ritorna un albero già potato dall'algoritmo dei Tagli Alfa–Beta.

```
  public GeneralTree getAlberoRisultato() {
    return p;
  }
```

3.3 Simulazione di un gioco generico

Per applicare l'algoritmo dei Tagli Alfa–Beta è necessario un gioco caratterizzato da regole. Per questo abbiamo realizzato la classe `GiocoGenericoConTagliAlfaBeta` che simula un gioco generico senza entrare nei dettagli di regole o meccanismi di valutazione di mosse. In appendice C è possibile visionare dettagliatamente il relativo codice sorgente.

La classe `GiocoGenericoConTagliAlfaBeta` estende la classe astratta `TagliAlfaBeta`

```
public class GiocoGenericoConTagliAlfaBeta extends TagliAlfaBeta
```

assumendosi l'onere di implementarne i metodi astratti `generaDescrittoriStatiFigli` e `valutaCostoMossa`.

3.3.1 Attributi

Questa classe non ha attributi: è semplicemente una classe per testare la `TagliAlfaBeta`.

3.3.2 Metodi

GiocoGenericoConTagliAlfaBeta

Il costruttore riceve come argomenti il descrittore dello stato iniziale `gsdStatoIniziale` del gioco e la profondità `profondita` a cui si desidera esplorare lo spazio degli stati.

```
public GiocoGenericoConTagliAlfaBeta( int profondita,
  GameStatusDescriptor gsdStatoIniziale ) {

  super( profondita, gsdStatoIniziale );
  }
```

Successivamente invoca il costruttore della super–classe `TagliAlfaBeta` passando i relativi argomenti.

main

Inizializza la profondità di esplorazione e viene creato un descrittore dello stato iniziale di gioco. Esso è creato in uno stato consistente con carattere maiuscolo `A`, da un etichetta simbolica $+\infty$, da un valore nullo, da una descrizione della mossa compiuta per raggiungerlo e dal giocatore che ha la mano, in questo caso MIN:

```
public static void main (String args[]) {
  int profondita = 10;

  /* Definizione dello stato iniziale del gioco */
  GameStatusDescriptor gsdStatoIniziale =
    new GameStatusDescriptor( new Character( 'A' ),
      GameStatusDescriptor.PIU_INFINITO, new Float( 0 ),
      "nessuna mossa fatta per raggiungere lo stato iniziale",
      GameStatusDescriptor.MIN );
```

Si crea l'oggetto `gioco` che provvederà a tagliare lo spazio degli stati, passandogli gli opportuni argomenti:

```
    GiocoGenericoConTagliAlfaBeta gioco =
      new GiocoGenericoConTagliAlfaBeta( profondita, gsdStatoIniziale );

    /* Visualizzo l'albero risultante dopo gli eventuali tagli */
    gioco.getAlberoRisultato().preOrder( gioco.getAlberoRisultato() ).print();
  }
```

Poiché tale oggetto ritorna un riferimento all'albero dello spazio degli stati, eventualmente potato, lo si visualizza tramite una visita in pre–order.

generaDescrittoriStatiFigli

Come precedentemente osservato questo metodo astratto deve essere ridefinito per poter applicare l'algoritmo dei Tagli Alfa–Beta. Ridefinito in modo che ricevuto un descrittore di stato `currentGSD` lo analizzi per capire quali siano le mosse possibili, ed in modo che ritorni una lista degli stati raggiungibili applicando la mossa opportuna:

```
public List generaDescrittoriStatiFigli( GameStatusDescriptor currentGSD )
```

Per semplicità non siamo entrati nel merito delle regole di alcun gioco. Semplicemente abbiamo creato la lista degli stati figli da ritornare. Ogni stato è costruito: con un carattere; con una etichetta simbolica `NON_MARCATO`; con una valutazione della mossa compiuta per raggiungerlo; con una descrizione della mossa compiuta per raggiungerlo; con il giocatore di mano:

```
/* Crea la lista degli stati figli, inizialmente vuota */
List statiFigli = new List( "lista degli stati figli" );

/* Se lo stato corrente è A */
if ( currentGSD.getGameStatus().equals( new Character( 'A' ) ) ) {

  /* Allora le mosse possibili sono quelle che porteranno
  * agli stati figli B e C */

  /* Genero lo stato figlio B */
  GameStatusDescriptor gsdFiglio =
    new GameStatusDescriptor( new Character( 'B' ),
      GameStatusDescriptor.NON_MARCATO,
      valutaCostoMossa( "mossa A ---> B" ),
      "mossa A ---> B",
      alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );

  /* Aggiungo lo stato figlio nella lista degli stati figli */
  statiFigli.insertAtBack( gsdFiglio );

  ...
  analogamente per il figlio C
  ...

  /* Ritorno la lista dei figli */
  return statiFigli;
}

...
tutti gli altri stati
...
```

valutaCostoMossa

Anche questo metodo astratto deve essere ridefinito per poter applicare l'algoritmo dei Tagli Alfa–Beta. Ridefinito in modo che ricevuto un descrizione della mossa da eseguire `moveToDone` la analizzi e ritorni il punteggio dello stato raggiunto applicando tale mossa.

Per semplicità non si entra nel merito del meccanismo di calcolo dei punteggi delle mosse: semplicemente si associa un punteggio alla mossa da effettuare:

```
public Float valutaCostoMossa( String moveToDone ) {

  if ( moveToDone.equals( "mossa D ---> H" ) ) return new Float( 28 );
  if ( moveToDone.equals( "mossa D ---> I" ) ) return new Float( 31 );

  ...
  tutte le altre mosse
  ...

  return new Float( 0 );
}
```

Per costruire rapidamente lo spazio degli stati relativi agli esercizi di verifica, abbiamo attribuito un punteggio non nullo solo alle mosse che portano a stati finali. Tutte le altre mosse hanno punteggio nullo.

Capitolo 4

Verifica dell'implementazione in Java

La correttezza dell'implementazione dell'algoritmo è stata verificata su esercizi di alcune prove scritte d'esame corredate di soluzione.

4.1 Esame del 25/06/2003

Si consideri l'albero di gioco in figura 4.1 dove i punteggi sono tutti dal punto di vista del primo giocatore. Supponendo che il primo giocatore sia MIN,

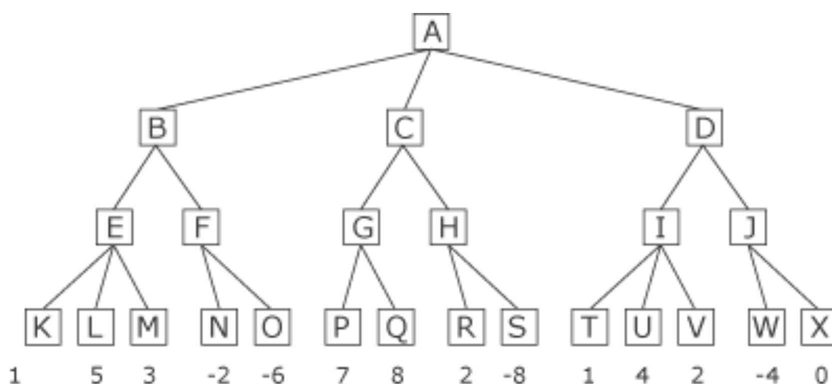

Figura 4.1: Spazio degli stati di gioco

quale mossa dovrebbe scegliere? Si risolva l'esercizio tramite l'algoritmo dei Tagli Alfa–Beta.

Per risolvere l'esercizio è stato necessario ridefinire i metodi astratti `generaDescrittoriStatiFigli` e `valutaCostoMossa`, come mostrato dettagliatamente in appendice C.1, che realizzano lo spazio degli stati di figura 4.1.

Il risultato dell'esecuzione dell'algoritmo è riportato in appendice D.1, di seguito riportiamo i risultati notevoli.

Gli stati tagliati risultano essere:

```
Tagliato il ramo che porta allo stato:
O
NON_MARCATO
-6.0
mossa F ---> O
MAX

Tagliato il ramo che porta allo stato:
H
NON_MARCATO
0.0
mossa C ---> H
MIN

Tagliato il ramo che porta allo stato:
J
NON_MARCATO
0.0
mossa D ---> J
MIN
```

confermando la soluzione data alla correzione della prova scritta e riportata in figura 4.2. Si noti che i nodi tagliati sono tutti etichettati con `NON_MARCATO` il che significa, come già detto in precedenza, che il nodo *non è stato esplorato.*

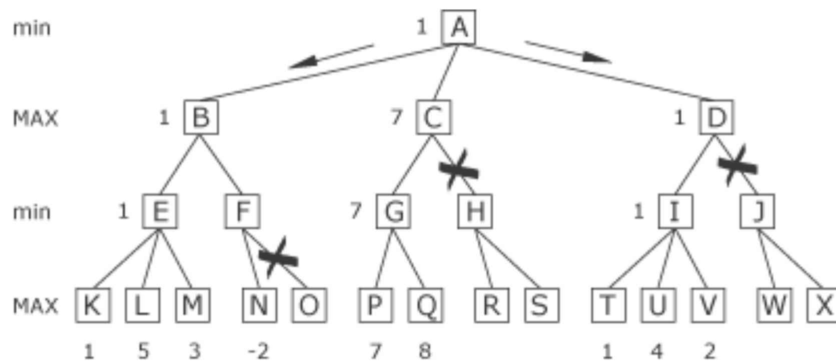

Figura 4.2: Spazio degli stati ridotto con i Tagli Alfa–Beta

La visita in pre–order dell'albero potato è correttamente risultata:

```
lista di visita in pre-order:
A
ASSEGNATO_VALORE
1.0
nessuna mossa fatta per raggiungere lo stato iniziale
MIN

B ... E ... K ... L ... M ... F ... N
```

```
C
ASSEGNATO_VALORE
7.0
mossa A ---> C
MAX

G  ... P  ... Q

D
ASSEGNATO_VALORE
1.0
mossa A ---> D
MAX

I ... T ... U ... V
```

Si noti che tutti gli altri nodi dell'albero sono etichettati con `ASSEGNATO_VALORE` il che significa, come già detto in precedenza, che ogni nodo è stato completamente esplorato. In particolare il campo `value` è il punteggio *definitivo* associato al nodo.

4.2 Esame del 18/03/2003

Dato l'albero di ricerca in figura 4.3 per un gioco a due giocatori, si mostri quale mossa selezionerà MAX secondo l'algoritmo dei Tagli Alfa–Beta.

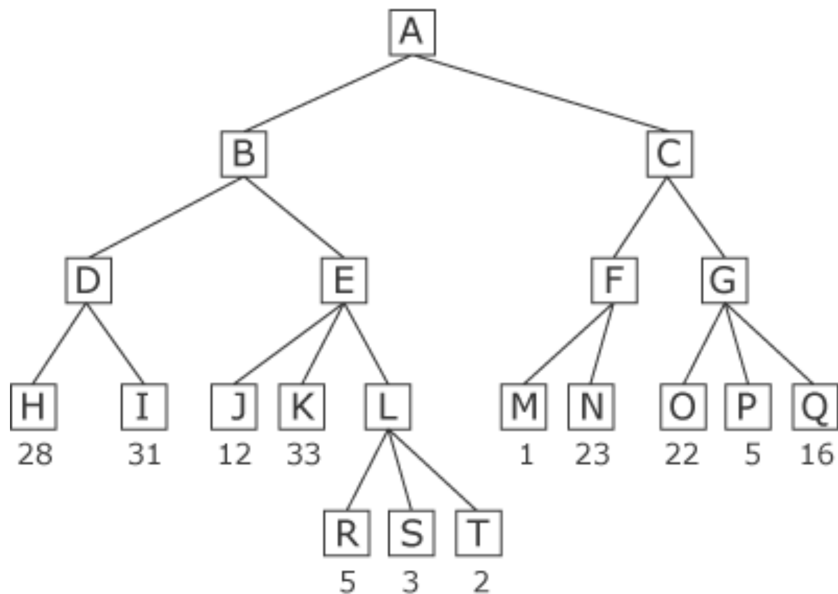

Figura 4.3: Spazio degli stati di gioco

Per risolvere l'esercizio è stato necessario ridefinire i metodi astratti `generaDescrittoriStatiFigli` e `valutaCostoMossa`, come mostrato dettagliatamente in appendice C.2, che realizzano lo spazio degli stati di figura 4.3.

Il risultato dell'esecuzione dell'algoritmo è riportato in appendice D.2, di seguito riportiamo i risultati notevoli.

Gli stati tagliati risultano essere:

```
Tagliato il ramo che porta allo stato:
L
NON_MARCATO
0.0
mossa E ---> L
MIN

Tagliato il ramo che porta allo stato:
G
NON_MARCATO
0.0
mossa C ---> G
MAX
```

confermando la soluzione data alla correzione della prova scritta e riportata in figura 4.4. Si noti che i nodi tagliati sono tutti etichettati con `NON_MARCATO` il che significa, come già detto in precedenza, che il nodo *non è stato esplorato*.

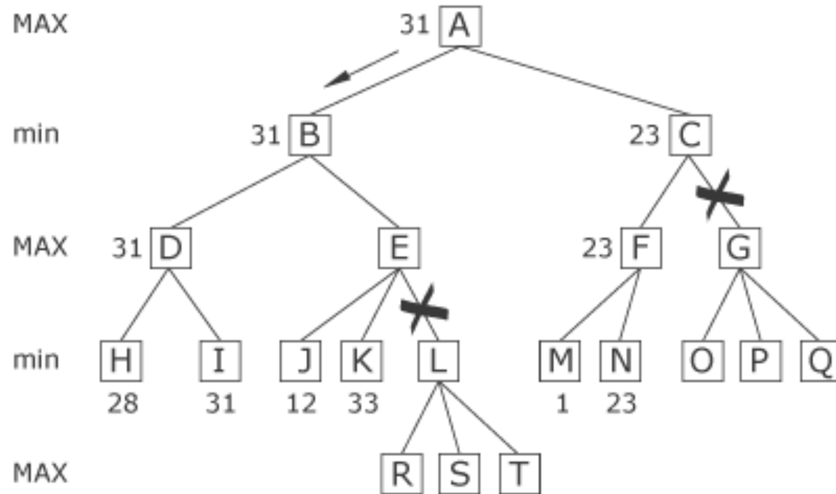

Figura 4.4: Spazio degli stati ridotto con i Tagli Alfa–Beta

La visita in pre–order dell'albero potato è correttamente risultata:

```
lista di visita in pre-order:
A
ASSEGNATO_VALORE
31.0
```

```
nessuna mossa fatta per raggiungere lo stato iniziale
MAX

B ... D ... H ... I ... E ... J ... K

C
ASSEGNATO_VALORE
23.0
mossa A ---> C
MIN

F ... M ... N
```

Si noti che tutti gli altri nodi dell'albero sono etichettati con `ASSEGNATO_VALORE` il che significa, come già detto in precedenza, che ogni nodo è stato completamente esplorato. In particolare il campo `value` è il punteggio *definitivo* associato al nodo.

4.3 Esame del 18/03/2003 - variante

Si consideri l'albero di ricerca in figura 4.3 e sia MIN, invece che MAX, il giocatore che inizialmente è di mano.

I metodi astratti `generaDescrittoriStatiFigli` e `valutaCostoMossa` sono sempre ridefiniti come mostrato in appendice D.2, che realizzano ancora lo spazio degli stati di figura 4.3.

Il risultato dell'esecuzione dell'algoritmo è invece riportato in appendice D.3, di seguito riportiamo i risultati notevoli.

Gli stati tagliati risultano essere:

```
Tagliato il ramo che porta allo stato:
K
NON_MARCATO
33.0
mossa E ---> K
MAX
```

Si noti che il nodo tagliato è etichettato con `NON_MARCATO` il che significa, come già detto in precedenza, che il nodo *non è stato esplorato*. La parte dedicata all'output della soluzione si limita a visualizzare il primo stato tagliato e non tutti gli stati tagliati (possibile miglioramento del programma).

La soluzione di figura 4.5 è correttamente calcolata, così come evidenziato dalla visita in pre–order dell'albero potato, che è risultata:

```
lista di visita in pre-order:
A
ASSEGNATO_VALORE
5.0
nessuna mossa fatta per raggiungere lo stato iniziale
MIN
```

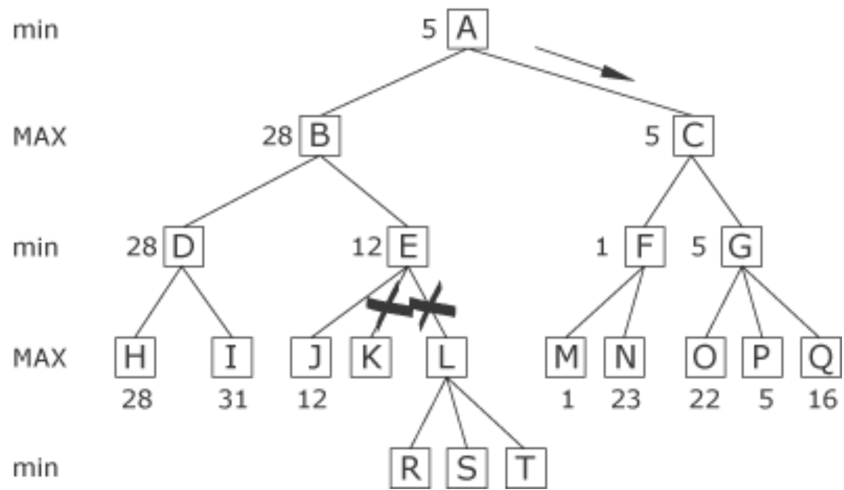

Figura 4.5: Spazio degli stati ridotto con i Tagli Alfa–Beta

```
B ... D ... H ... I ... E ... J

C
ASSEGNATO_VALORE
5.0
mossa A ---> C
MAX

F ... M ... N ... G ... O ... P
```

Si noti che tutti gli altri nodi dell'albero sono etichettati con `ASSEGNATO_VALORE` il che significa, come già detto in precedenza, che ogni nodo è stato completamente esplorato. In particolare il campo `value` è il punteggio *definitivo* associato al nodo.

Conclusioni

Gli esempi scelti per la verifica dell'algoritmo dei Tagli Alfa–Beta coprono una casistica che ha permesso di testare con successo la correttezza dell'implementazione in tutte le sue parti.

Possibili miglioramenti potrebbero adottarsi relativamente alla parte di codice che cura l'output dei risultati. Per esempio si potrebbe porre in una lista `statiTagliati` tutti gli stati potati dall'albero dello spazio degli stati, e scrivere un metodo `getStatiTagliati` per ritornare un riferimento a tale lista in modo da poterla visualizzare agevolmente.

Possibili miglioramenti potrebbero essere apportati anche alla parte dedicata alla definizione del gioco generico tramite i metodi `generaStatiFigli` e `valutaCostoMossa`. Tuttavia è stato più breve scrivere il codice di tali metodi invece che progettare una piccola applicazione apposita. Tale miglioramento non è poi così necessario come potrebbe apparire ad una prima analisi, considerando anche che chi vorrà applicare i Tagli Alfa–Beta lo farà su un particolare tipo di gioco, e non su uno generico creato solamente per testare la classe `TagliAlfaBeta`.

Infine un miglioramento, dal punto di vista dell'Ingegneria del Software, potrebbe essere quello di utilizzare oggetti, quali liste ed alberi, appartenenti a classi *standard* più efficienti e sicure rispetto a quelle da noi utilizzate. Queste ultime offrono i "classici" comportamenti di alberi e liste e per completezza ne riportiamo il codice in appendice E.

Appendice A

Classe GameStatusDescriptor

```
/*
 * GameStatusDescriptor
 * Copyright (C) 2004 Tarin Gamberini
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this program; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */


/*
 * GameStatusDescriptor
 * Copyright (C) 2004 Tarin Gamberini
 *
 * Questo programma è software libero; è lecito redistribuirlo o
 * modificarlo secondo i termini della Licenza Pubblica Generica
 * GNU come è pubblicata dalla Free Software Foundation; o la
 * versione 2 della licenza o (a propria scelta) una versione
 * successiva.
 *
 * Questo programma è distribuito nella speranza che sia utile, ma
 * SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di
 * NEGOZIABILITÀ o di APPLICABILITÀ PER UN PARTICOLARE SCOPO. Si
 * veda la Licenza Pubblica Generica GNU per avere maggiori
 * dettagli.
 *
 * Questo programma deve essere distribuito assieme ad una copia
 * della Licenza Pubblica Generica GNU; in caso contrario, se ne
 * può ottenere una scrivendo alla Free Software Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 */
```

```
public class GameStatusDescriptor {

    /* Valori costanti assunti dal campo label */
    public static String NON_MARCATO = "NON_MARCATO";
    public static String ASSEGNATO_VALORE = "ASSEGNATO_VALORE";
    public static String PIU_INFINITO = "PIU_INFINITO";
    public static String MENO_INFINITO = "MENO_INFINITO";
    public static String MAGGIORE_O_UGUALE = "MAGGIORE_O_UGUALE";
    public static String MINORE_O_UGUALE = "MINORE_O_UGUALE";

    /* Valori costanti assunti dal campo hasToMove */
    public static String MAX = "MAX";
    public static String MIN = "MIN";

    /* Riferimento allo stato di gioco corrente */
    private Object gameStatus;
    /* Etichetta simbolica */
    private String label;
    /* Valore del nodo associato allo stato gameStatus */
    private Float value;
    /* Mossa compiuta per raggiungere lo stato corrente */
    private String moveDone;
    /* Giocatore che ha la mano e che quindi deve compiere la mossa. */
    private String hasToMove;

    /** Creates new GameStatusDescriptor settato ad uno stato vuoto. */
    public GameStatusDescriptor() {
        this( new String( "nessuno stato corrente" ), NON_MARCATO, new Float( 0 ),
        "nessuna mossa fatta", "nessun giocatore deve muovere" );
    }

    /** Creates new GameStatusDescriptor in uno stato specificato dagli argomenti
     * passati al metodo. */
    public GameStatusDescriptor( Object gameStatusToSet, String labelToSet,
    Float valueToSet, String moveDoneToSet, String hasToMoveToSet ) {
        setGameStatus( gameStatusToSet );
        setLabel( labelToSet );
        setValue( valueToSet );
        setMoveDone( moveDoneToSet );
        setHasToMove( hasToMoveToSet );
    }

    /** Setta il riferimento ad un oggetto che descrive lo stato. */
    public void setGameStatus( Object gameStatusToSet ) {
        gameStatus = gameStatusToSet;
    }

    /** Setta l'etichetta associata allo stato. */
    public void setLabel( String labelToSet ) {
        label = labelToSet;
    }

    /** Setta il valore associato allo stato corrente. */
    public void setValue( Float valueToSet ) {
        value = valueToSet;
    }

    /** Setta la mossa compiuta per raggiungere lo stato corrente. */
    public void setMoveDone( String moveDoneToSet ) {
        moveDone = moveDoneToSet;
    }
```

```
    /** Setta il nome del giocatore che deve muovere. */
    public void setHasToMove( String hasToMoveToSet ) {
        hasToMove = hasToMoveToSet;
    }

    /** Ritorna il riferimento ad un oggetto che descrive lo stato. */
    public Object getGameStatus() {
        return gameStatus;
    }

    /** Ritorna l'etichetta associata allo stato. */
    public String getLabel() {
        return label;
    }

    /** Ritorna il valore associato allo stato corrente. */
    public Float getValue() {
        return value;
    }

    /** Ritorna la mossa compiuta per raggiungere lo stato corrente. */
    public String getMoveDone() {
        return moveDone;
    }

    /** Ritorna il nome del giocatore che deve muovere. */
    public String getHasToMove() {
        return hasToMove;
    }

    /** Ritorna una stringa rappresentativa dello stato. */
    public String toString() {
        return ( "\n" +
            gameStatus.toString() + "\n" +
            label.toString() + "\n" +
            value.toString()  + "\n" +
            moveDone.toString()  + "\n" +
            hasToMove.toString() + "\n" );
    }

}
```

Appendice B

Classe astratta TagliAlfaBeta


```
/*
 * TagliAlfaBeta
 * Copyright (C) 2004 Tarin Gamberini
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this program; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */
```

```
/*
 * TagliAlfaBeta
 * Copyright (C) 2004 Tarin Gamberini
 *
 * Questo programma è software libero; è lecito redistribuirlo o
 * modificarlo secondo i termini della Licenza Pubblica Generica
 * GNU come è pubblicata dalla Free Software Foundation; o la
 * versione 2 della licenza o (a propria scelta) una versione
 * successiva.
 *
 * Questo programma è distribuito nella speranza che sia utile, ma
 * SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di
 * NEGOZIABILITÀ o di APPLICABILITÀ PER UN PARTICOLARE SCOPO. Si
 * veda la Licenza Pubblica Generica GNU per avere maggiori
 * dettagli.
 *
 * Questo programma deve essere distribuito assieme ad una copia
 * della Licenza Pubblica Generica GNU; in caso contrario, se ne
 * può ottenere una scrivendo alla Free Software Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 */
```

```
public abstract class TagliAlfaBeta {

    /** Riceve il descrittore di stato di gioco currentGSD,
     * lo analizza per scoprire le mosse
     * possibili, e ritorna una lista di descrittori di stato futuro. */
    public abstract List generaDescrittoriStatiFigli( GameStatusDescriptor currentGSD );

    /** Valuta il costo della mossa specificata nella stringa moveToDone. */
    public abstract Float valutaCostoMossa( String moveToDone );



    /* Albero rappresentante lo spazio degli stati del gioco. */
    private GeneralTree p;

    /* Profondità massima a cui scendere nell'esplorazione dell'albero. */
    private int profonditaMassima;

    /** Crea un nuovo TagliAlfaBeta */
    public TagliAlfaBeta( int profonditaMassimaFissata,
                          GameStatusDescriptor gsdStatoIniziale ) {

        /* Setta la profondità massima a cui scendere nell'esplorazione dell'albero. */
        profonditaMassima = profonditaMassimaFissata;

        /* Crea l'albero dello spazio degli stati di gioco inizializzandolo
         * allo stato iniziale*/
        p = new GeneralTree( "albero dello spazio degli stati", gsdStatoIniziale );

        /* esploraConTagliAlfaBeta( p , padreDip , profonditaCorrente ) */
        esploraConTagliAlfaBeta( p , null , 0 );
    }

    /* Ritorna l'albero dello spazio degli stati di gioco dopo l'applicazione
     * dell'algoritmo dei Tagli Alfa Beta. */
    public GeneralTree getAlberoRisultato() {
        return p;
    }

    /** Esplora l'albero con strategia depth first effettuando eventuali
     * tagli alfa e beta. */
    private void esploraConTagliAlfaBeta( GeneralTree p, GeneralTree padreDiP,
                                          int profonditaCorrente ) {

        /* Etichetta un nodo con valore simbolico PIU_INFINITO o MENO_INFINITO. */
        etichettaConValoreSimbolico( p );

        /* Si valuta l'esistenza di possibili stati figli */
        List DescrittoriStatiFigliPossibili =
        generaDescrittoriStatiFigli( ( GameStatusDescriptor ) p.getRoot().getObject() );

        /* Se non è stata raggiunta la massima profondità ed esistono stati figli */
        if ( profonditaCorrente < profonditaMassima &&
             !DescrittoriStatiFigliPossibili.isEmpty() ) {

                /* Dallo stato corrente si generano gli stati figli */
                p.appendListOfSons( p, DescrittoriStatiFigliPossibili );

                /* Si esplorano ricorsivamente gli stati figli */
                ListNode pDescrittoreFiglioCorrente = p.getRoot().getNexts().getFirst();
                /* Posizione del pDescrittoreFiglioCorrente nella lista degli stati figli. */
```

```
            int posizioneDescrittoreFiglioCorrente = 1;
            while ( pDescrittoreFiglioCorrente != null ) {
                esploraConTagliAlfaBeta(
                  ( GeneralTree ) pDescrittoreFiglioCorrente.getObject(),
                  p, profonditaCorrente + 1 );
                aggiornaValore(
                  p, ( GeneralTree ) pDescrittoreFiglioCorrente.getObject() );
                if ( valutaSeTagliare( p, padreDiP ) ) {
                    System.out.println( "Tagliato il ramo che porta allo stato:" +
                    ( ( GeneralTree ) DescrittoreFiglioCorrente.getNext().getObject() )
                        .getRoot().getObject().toString() );
                    /* taglia tutti i figli successivi a quello corrente */
                    p.getRoot().getNexts().cutFromPosition(
                                  posizioneDescrittoreFiglioCorrente + 1 );
                }
                pDescrittoreFiglioCorrente = pDescrittoreFiglioCorrente.getNext();
                posizioneDescrittoreFiglioCorrente++;
            }

            /* Esplorati tutti i figli di p occorre rendere il valore
             * provvisorio di p MAGGIORE_O_UGUALE o MINORE_O_UGUALE
             * effettivo, settandolo ad ASSEGNATO_VALORE */
            ( ( GameStatusDescriptor ) p.getRoot().getObject() )
            .setLabel( GameStatusDescriptor.ASSEGNATO_VALORE );
    }
    else
        /* Se altrimenti
         * o  è stata raggiunta la massima profondità di esplorazione e (anche se)
         *    esistono stati figli
         * o  non è stata raggiunta la massima profondità di esplorazione e (ma)
         *    non esistono stati figli
         * o  è stata raggiunta la massima profondità di esplorazione e
         *    non esistono stati figli
         * è impossibile esplorare ulteriormente l'albero degli stati di gioco
         * scendendo più in profondità */

        /* Il valore provvisorio assegnato allo stato viene reso definitivo */
        ( ( GameStatusDescriptor ) p.getRoot().getObject() )
        .setLabel( GameStatusDescriptor.ASSEGNATO_VALORE );
}

/** Etichetta un nodo con valore simbolico PIU_INFINITO o MENO_INFINITO. */
private void etichettaConValoreSimbolico(GeneralTree p) {
    /* Se è di mano MAX */
    if ( ( ( GameStatusDescriptor ) p.getRoot().getObject() )
    .getHasToMove().equals( GameStatusDescriptor.MAX ) )
        /* Etichetta il nodo con valore simbolico MENO_INFINITO */
        ( ( GameStatusDescriptor ) p.getRoot().getObject() )
        .setLabel( GameStatusDescriptor.MENO_INFINITO );
    /* Altrimenti è di mano MIN */
    else
        /* Etichetta il nodo con valore simbolico PIU_INFINITO */
        ( ( GameStatusDescriptor ) p.getRoot().getObject() )
        .setLabel( GameStatusDescriptor.PIU_INFINITO );
}

/** Aggiorna il valore del nodo padre pFather al massimo (se in pFather è di
 * mano MAX) o minimo (se in pFather è di mano MIN) fra il valore del padre
 * e quello di pSon, figlio di pFather. */
private void aggiornaValore(GeneralTree pFather, GeneralTree pSon) {

    GameStatusDescriptor pFatherGSD =
```

```
        ( GameStatusDescriptor ) pFather.getRoot().getObject();
      GameStatusDescriptor pSonGSD =
        ( GameStatusDescriptor ) pSon.getRoot().getObject();

      /* Se nel nodo padre è di mano MAX */
      if ( pFatherGSD.getHasToMove().equals( GameStatusDescriptor.MAX ) )
          /* Se il nodo padre era etichettato con valore simbolico MENO_INFINITO */
          if ( pFatherGSD.getLabel().equals( GameStatusDescriptor.MENO_INFINITO ) ) {
              /* Nel padre viene settato un massimo provvisorio, inizialmente
               * uguale al valore del figlio */
              pFatherGSD.setLabel( GameStatusDescriptor.MAGGIORE_O_UGUALE );
              pFatherGSD.setValue( pSonGSD.getValue() );
          }
          else /* Altrimenti il nodo padre è etichettato MAGGIORE_O_UGUALE ad un
                * valore numerico provvisorio */
              /* Nel padre si setta un nuovo massimo provvisorio fra il valore
               * del padre e del figlio */
              pFatherGSD.setValue( massimo( pFatherGSD.getValue(), pSonGSD.getValue() ) );
      else /* Altrimenti nel nodo padre è di mano MIN */
          /* Se il nodo padre era etichettato con valore simbolico PIU_INFINITO */
          if ( pFatherGSD.getLabel().equals( GameStatusDescriptor.PIU_INFINITO ) ) {
              /* Nel padre viene settato un minimo provvisorio, inizialmente
               * uguale al valore del figlio */
              pFatherGSD.setLabel( GameStatusDescriptor.MINORE_O_UGUALE );
              pFatherGSD.setValue( pSonGSD.getValue() );
          }
          else /* Altrimenti il nodo padre è etichettato MINORE_O_UGUALE ad un
                * valore numerico provvisorio */
              /* Nel padre si setta un nuovo minimo provvisorio fra il valore
               * del padre e del figlio */
              pFatherGSD.setValue( minimo( pFatherGSD.getValue(), pSonGSD.getValue() ) );
  }

  /** Ritorna il minimo fra a e b */
  private Float minimo( Float a, Float b ) {
      if ( a.compareTo( b ) <= 0 )
          return a;
      else
          return b;
  }

  /** Ritorna il massimo fra a e b */
  private Float massimo( Float a, Float b ) {
      if ( a.compareTo( b ) >= 0 )
          return a;
      else
          return b;
  }

  /** Ritorna true se è il caso di effettuare un taglio */
  private boolean valutaSeTagliare( GeneralTree p, GeneralTree pFather ) {

      if ( pFather != null ) {

          GameStatusDescriptor pGSD =
            ( GameStatusDescriptor ) p.getRoot().getObject();
          GameStatusDescriptor pFatherGSD =
            ( GameStatusDescriptor ) pFather.getRoot().getObject();

          /* Se i valori provvisori fra padre e figlio individuano insiemi ad
           * intersezione nulla è il caso di tagliare */
          if ( pFatherGSD.getLabel().equals( GameStatusDescriptor.MAGGIORE_O_UGUALE ) &&
```

```
                    pGSD.getLabel().equals( GameStatusDescriptor.MINORE_O_UGUALE ) ) {

                        if ( pFatherGSD.getValue().compareTo( pGSD.getValue() ) >= 0 )
                            return true;
            }
            else
                if ( pFatherGSD.getLabel().equals( GameStatusDescriptor.MINORE_O_UGUALE )&&
                     pGSD.getLabel().equals( GameStatusDescriptor.MAGGIORE_O_UGUALE ) ) {

                        if ( pFatherGSD.getValue().compareTo( pGSD.getValue() ) <= 0 )
                            return true;
            }

            /* Se i valori provvisori fra padre e figlio individuano insiemi ad
             * intersezione non nulla allora non è il caso di tagliare, ma
             * occorre lasciar proseguire l'algoritmo */
            return false;
        }

        /* Se il padre è null allora stiamo analizzando la radice dell'albero
         * e quindi non esistono fratelli da tagliare */
        return false;
    }

}
```

Appendice C

Classe GiocoGenericoConTagliAlfaBeta

La classe che abbiamo scritto per descrivere un gioco generico presenta un costruttore ed un metodo `main` indipendenti dal tipo di gioco. Il costruttore richiama quello della classe `TagliAlfaBeta` mentre il metodo `main` inizializza le variabili per verificare con l'algoritmo dei Tagli Alfa–Beta.

```
public class GiocoGenericoConTagliAlfaBeta extends TagliAlfaBeta {

    /** Creates new GiocoGenericoConTagliAlfaBeta */
    public GiocoGenericoConTagliAlfaBeta
     int profondita, GameStatusDescriptor gsdStatoIniziale ) {
        super( profondita, gsdStatoIniziale );
    }

    public static void main (String args[]) {

        /* Profondità massima di esplorazione. La radice ha profondità 0 */
        int profondita = 10;

        /* Definizione dello stato iniziale del gioco */
        GameStatusDescriptor gsdStatoIniziale =
            new GameStatusDescriptor( new Character( 'A' ),
                GameStatusDescriptor.PIU_INFINITO, new Float( 0 ),
                "nessuna mossa fatta per raggiungere lo stato iniziale",
                GameStatusDescriptor.MIN );

        GiocoGenericoConTagliAlfaBeta gioco =
          new GiocoGenericoConTagliAlfaBeta( profondita, gsdStatoIniziale );

        /* Visualizzo l'albero risultante dopo gli eventuali tagli */
        gioco.getAlberoRisultato().preOrder( gioco.getAlberoRisultato() ).print();
    }
```

Il codice che invece dipende dal particolare gioco è quello dei metodi `generaStatiFigli` e `valutaCostoMossa` qui di seguito riportati in due versioni: una per il gioco proposto nell'esame del 25/06/2003 e l'altra per il gioco proposto nell'esame del 18/03/2003.

C.1 Esame del 25/06/2003

C.1.1 Metodo generaStatiFigli

```
/** Riceve il descrittore di stato di gioco currentGSD, lo analizza per
 * scoprire le mosse possibili, e ritorna una lista di descrittori di
 * stato futuro. */
public List generaStatiFigli(GeneralTree p) {
    /* Per semplicità non si entra nel merito delle regole di alcun gioco.
     * Semplicemente si costruisce la lista degli stati figli
     * raggiungibili dallo stato corrente applicando le mosse possibili. */

    /* Crea la lista degli stati figli, inizialmente vuota */
    List statiFigli = new List( "lista degli stati figli" );

    /* Prelevo il descrittore corrente dello stato di gioco  */
    GameStatusDescriptor currentGSD =
      ( GameStatusDescriptor )p.getRoot().getObject();

    /* Se lo stato corrente è A */
    if ( currentGSD.getGameStatus().equals( new Character( 'A' ) ) ) {
        /* Allora le mosse possibili sono quelle che porteranno
         * agli stati figli B, C e D */

        /* Genero lo stato figlio B */
        GameStatusDescriptor gsdFiglio =
            new GameStatusDescriptor( new Character( 'B' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa A ---> B" ),
            "mossa A ---> B",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio C */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'C' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa A ---> C" ),
            "mossa A ---> C",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio D */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'D' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa A ---> D" ),
            "mossa A ---> D",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }

    /* Se lo stato corrente è B */
    if ( currentGSD.getGameStatus().equals( new Character( 'B' ) ) ) {
        /* Allora le mosse possibili sono quelle che porteranno
```

```
             * agli stati figli E e F */

            /* Genero lo stato figlio E */
            GameStatusDescriptor gsdFiglio =
                new GameStatusDescriptor( new Character( 'E' ),
                GameStatusDescriptor.NON_MARCATO,
                valutaCostoMossa( "mossa B ---> E" ),
                "mossa B ---> E",
                alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
            /* Aggiungo lo stato figlio nella lista degli stati figli */
            statiFigli.insertAtBack( gsdFiglio );

            /* Genero lo stato figlio F */
            gsdFiglio =
                new GameStatusDescriptor( new Character( 'F' ),
                GameStatusDescriptor.NON_MARCATO,
                valutaCostoMossa( "mossa B ---> F" ),
                "mossa B ---> F",
                alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
            /* Aggiungo lo stato figlio nella lista degli stati figli */
            statiFigli.insertAtBack( gsdFiglio );

            /* Ritorno la lista dei figli */
            return statiFigli;
        }

        /* Se lo stato corrente è C */
        if ( currentGSD.getGameStatus().equals( new Character( 'C' ) ) ) {
            /* Allora le mosse possibili sono quelle che porteranno
             * agli stati figli G e H */

            /* Genero lo stato figlio G */
            GameStatusDescriptor gsdFiglio =
                new GameStatusDescriptor( new Character( 'G' ),
                GameStatusDescriptor.NON_MARCATO,
                valutaCostoMossa( "mossa C ---> G" ),
                "mossa C ---> G",
                alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
            /* Aggiungo lo stato figlio nella lista degli stati figli */
            statiFigli.insertAtBack( gsdFiglio );

            /* Genero lo stato figlio H */
            gsdFiglio =
                new GameStatusDescriptor( new Character( 'H' ),
                GameStatusDescriptor.NON_MARCATO,
                valutaCostoMossa( "mossa C ---> H" ),
                "mossa C ---> H",
                alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
            /* Aggiungo lo stato figlio nella lista degli stati figli */
            statiFigli.insertAtBack( gsdFiglio );

            /* Ritorno la lista dei figli */
            return statiFigli;
        }

        /* Se lo stato corrente è D */
        if ( currentGSD.getGameStatus().equals( new Character( 'D' ) ) ) {
            /* Allora le mosse possibili sono quelle che porteranno
             * agli stati figli I e J */

            /* Genero lo stato figlio I */
            GameStatusDescriptor gsdFiglio =
```

```
            new GameStatusDescriptor( new Character( 'I' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa D ---> I" ),
            "mossa D ---> I",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio J */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'J' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa D ---> J" ),
            "mossa D ---> J",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }

    /* Se lo stato corrente è E */
    if ( currentGSD.getGameStatus().equals( new Character( 'E' ) ) ) {
        /* Allora le mosse possibili sono quelle che porteranno
         * agli stati figli K, L e M */

        /* Genero lo stato figlio K */
        GameStatusDescriptor gsdFiglio =
            new GameStatusDescriptor( new Character( 'K' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa E ---> K" ),
            "mossa E ---> K",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio L */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'L' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa E ---> L" ),
            "mossa E ---> L",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio M */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'M' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa E ---> M" ),
            "mossa E ---> M",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }

    /* Se lo stato corrente è F */
```

```
if ( currentGSD.getGameStatus().equals( new Character( 'F' ) ) ) {
    /* Allora le mosse possibili sono quelle che porteranno
     * agli stati figli N e O */

    /* Genero lo stato figlio N */
    GameStatusDescriptor gsdFiglio =
        new GameStatusDescriptor( new Character( 'N' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa F ---> N" ),
        "mossa F ---> N",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Genero lo stato figlio O */
    gsdFiglio =
        new GameStatusDescriptor( new Character( 'O' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa F ---> O" ),
        "mossa F ---> O",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Ritorno la lista dei figli */
    return statiFigli;
}

/* Se lo stato corrente è G */
if ( currentGSD.getGameStatus().equals( new Character( 'G' ) ) ) {
    /* Allora le mosse possibili sono quelle che porteranno
     * agli stati figli P e Q */

    /* Genero lo stato figlio P */
    GameStatusDescriptor gsdFiglio =
        new GameStatusDescriptor( new Character( 'P' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa G ---> P" ),
        "mossa G ---> P",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Genero lo stato figlio Q */
    gsdFiglio =
        new GameStatusDescriptor( new Character( 'Q' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa G ---> Q" ),
        "mossa G ---> Q",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Ritorno la lista dei figli */
    return statiFigli;
}

/* Se lo stato corrente è H */
if ( currentGSD.getGameStatus().equals( new Character( 'H' ) ) ) {
    /* Allora le mosse possibili sono quelle che porteranno
     * agli stati figli R e S */
```

```
        /* Genero lo stato figlio R */
        GameStatusDescriptor gsdFiglio =
            new GameStatusDescriptor( new Character( 'R' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa H ---> R" ),
            "mossa H ---> R",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio S */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'S' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa H ---> S" ),
            "mossa H ---> S",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }

    /* Se lo stato corrente è I */
    if ( currentGSD.getGameStatus().equals( new Character( 'I' ) ) ) {
        /* Allora le mosse possibili sono quelle che porteranno
         * agli stati figli T, U e V */

        /* Genero lo stato figlio T */
        GameStatusDescriptor gsdFiglio =
            new GameStatusDescriptor( new Character( 'T' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa I ---> T" ),
            "mossa I ---> T",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio U */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'U' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa I ---> U" ),
            "mossa I ---> U",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio V */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'V' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa I ---> V" ),
            "mossa I ---> V",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }
```

```
        /* Se lo stato corrente è J */
        if ( currentGSD.getGameStatus().equals( new Character( 'J' ) ) ) {
            /* Allora le mosse possibili sono quelle che porteranno
             * agli stati figli W e X */

            /* Genero lo stato figlio W */
            GameStatusDescriptor gsdFiglio =
                new GameStatusDescriptor( new Character( 'W' ),
                GameStatusDescriptor.NON_MARCATO,
                valutaCostoMossa( "mossa J ---> W" ),
                "mossa J ---> W",
                alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
            /* Aggiungo lo stato figlio nella lista degli stati figli */
            statiFigli.insertAtBack( gsdFiglio );

            /* Genero lo stato figlio X */
            gsdFiglio =
                new GameStatusDescriptor( new Character( 'X' ),
                GameStatusDescriptor.NON_MARCATO,
                valutaCostoMossa( "mossa J ---> X" ),
                "mossa J ---> X",
                alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
            /* Aggiungo lo stato figlio nella lista degli stati figli */
            statiFigli.insertAtBack( gsdFiglio );

            /* Ritorno la lista dei figli */
            return statiFigli;
        }

        /* Se lo stato corrente non è stato individuato nelle casistiche
         * precedenti, per la generazione di stati figli, significa che lo
         * stato corrente è uno stato finale: uno stato in cui non
         * sono possibili ulteriori mosse.
         * Per questo si ritorna una lista vuota. */
        statiFigli.setName( statiFigli.getName() + " vuota" );
        return statiFigli;
    }
```

C.1.2 Metodo valutaCostoMossa

```
    public Float valutaCostoMossa( String moveToDone ) {
        /* Per semplicità non si entra nel merito del meccanismo di calcolo
         * dei punteggi delle mosse.
         * Semplicemente si associa un punteggio alla mossa da effettuare. */

        if ( moveToDone.equals( "mossa E ---> K" ) ) return new Float( 1 );
        if ( moveToDone.equals( "mossa E ---> L" ) ) return new Float( 5 );
        if ( moveToDone.equals( "mossa E ---> M" ) ) return new Float( 3 );

        if ( moveToDone.equals( "mossa F ---> N" ) ) return new Float( -2 );
        if ( moveToDone.equals( "mossa F ---> O" ) ) return new Float( -6 );

        if ( moveToDone.equals( "mossa G ---> P" ) ) return new Float( 7 );
        if ( moveToDone.equals( "mossa G ---> Q" ) ) return new Float( 8 );

        if ( moveToDone.equals( "mossa H ---> R" ) ) return new Float( 2 );
        if ( moveToDone.equals( "mossa H ---> S" ) ) return new Float( -8 );

        if ( moveToDone.equals( "mossa I ---> T" ) ) return new Float( 1 );
```

```
        if ( moveToDone.equals( "mossa I ---> U" ) ) return new Float( 4 );
        if ( moveToDone.equals( "mossa I ---> V" ) ) return new Float( 2 );

        if ( moveToDone.equals( "mossa J ---> W" ) ) return new Float( -4 );
        if ( moveToDone.equals( "mossa J ---> X" ) ) return new Float( 0 );

        return new Float( 0 );
    }
```

C.2 Esame del 18/03/2003

C.2.1 Metodo generaStatiFigli

```
    /** Riceve il descrittore di stato di gioco currentGSD, lo analizza per
     * scoprire le mosse possibili, e ritorna una lista di descrittori di
     * stato futuro. */
    public List generaStatiFigli(GeneralTree p) {
        /* Per semplicità non si entra nel merito delle regole di alcun gioco.
         * Semplicemente si costruisce la lista degli stati figli
         * raggiungibili dallo stato corrente applicando le mosse possibili. */

        /* Crea la lista degli stati figli, inizialmente vuota */
        List statiFigli = new List( "lista degli stati figli" );

        /* Prelevo il descrittore corrente dello stato di gioco  */
        GameStatusDescriptor currentGSD =
          ( GameStatusDescriptor )p.getRoot().getObject();

        /* Se lo stato corrente è A */
        if ( currentGSD.getGameStatus().equals( new Character( 'A' ) ) ) {
            /* Allora le mosse possibili sono quelle che porteranno
             * agli stati figli B e C */

            /* Genero lo stato figlio B */
            GameStatusDescriptor gsdFiglio =
                new GameStatusDescriptor( new Character( 'B' ),
                GameStatusDescriptor.NON_MARCATO,
                valutaCostoMossa( "mossa A ---> B" ),
                "mossa A ---> B",
                alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
            /* Aggiungo lo stato figlio nella lista degli stati figli */
            statiFigli.insertAtBack( gsdFiglio );

            /* Genero lo stato figlio C */
            gsdFiglio =
                new GameStatusDescriptor( new Character( 'C' ),
                GameStatusDescriptor.NON_MARCATO,
                valutaCostoMossa( "mossa A ---> C" ),
                "mossa A ---> C",
                alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
            /* Aggiungo lo stato figlio nella lista degli stati figli */
            statiFigli.insertAtBack( gsdFiglio );

            /* Ritorno la lista dei figli */
            return statiFigli;
        }

        /* Se lo stato corrente è B */
```

```
if ( currentGSD.getGameStatus().equals( new Character( 'B' ) ) ) {
    /* Allora le mosse possibili sono quelle che porteranno
     * agli stati figli D e E */

    /* Genero lo stato figlio D */
    GameStatusDescriptor gsdFiglio =
        new GameStatusDescriptor( new Character( 'D' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa B ---> D" ),
        "mossa B ---> D",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Genero lo stato figlio E */
    gsdFiglio =
        new GameStatusDescriptor( new Character( 'E' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa B ---> E" ),
        "mossa B ---> E",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Ritorno la lista dei figli */
    return statiFigli;
}

/* Se lo stato corrente è C */
if ( currentGSD.getGameStatus().equals( new Character( 'C' ) ) ) {
    /* Allora le mosse possibili sono quelle che porteranno
     * agli stati figli F e G */

    /* Genero lo stato figlio F */
    GameStatusDescriptor gsdFiglio =
        new GameStatusDescriptor( new Character( 'F' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa C ---> F" ),
        "mossa C ---> F",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Genero lo stato figlio G */
    gsdFiglio =
        new GameStatusDescriptor( new Character( 'G' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa C ---> G" ),
        "mossa C ---> G",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Ritorno la lista dei figli */
    return statiFigli;
}

/* Se lo stato corrente è D */
if ( currentGSD.getGameStatus().equals( new Character( 'D' ) ) ) {
    /* Allora le mosse possibili sono quelle che porteranno
     * agli stati figli H e I */
```

```
        /* Genero lo stato figlio H */
        GameStatusDescriptor gsdFiglio =
            new GameStatusDescriptor( new Character( 'H' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa D ---> H" ),
            "mossa D ---> H",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio I */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'I' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa D ---> I" ),
            "mossa D ---> I",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }

    /* Se lo stato corrente è E */
    if ( currentGSD.getGameStatus().equals( new Character( 'E' ) ) ) {
        /* Allora le mosse possibili sono quelle che porteranno
         * agli stati figli J, K e L */

        /* Genero lo stato figlio J */
        GameStatusDescriptor gsdFiglio =
            new GameStatusDescriptor( new Character( 'J' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa E ---> J" ),
            "mossa E ---> J",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio K */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'K' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa E ---> K" ),
            "mossa E ---> K",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio L */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'L' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa E ---> L" ),
            "mossa E ---> L",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }
```

```
/* Se lo stato corrente è F */
if ( currentGSD.getGameStatus().equals( new Character( 'F' ) ) ) {
    /* Allora le mosse possibili sono quelle che porteranno
     * agli stati figli M e N */

    /* Genero lo stato figlio M */
    GameStatusDescriptor gsdFiglio =
        new GameStatusDescriptor( new Character( 'M' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa F ---> M" ),
        "mossa F ---> M",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Genero lo stato figlio N */
    gsdFiglio =
        new GameStatusDescriptor( new Character( 'N' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa F ---> N" ),
        "mossa F ---> N",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Ritorno la lista dei figli */
    return statiFigli;
}

/* Se lo stato corrente è G */
if ( currentGSD.getGameStatus().equals( new Character( 'G' ) ) ) {
    /* Allora le mosse possibili sono quelle che porteranno
     * agli stati figli O, P e Q */

    /* Genero lo stato figlio O */
    GameStatusDescriptor gsdFiglio =
        new GameStatusDescriptor( new Character( 'O' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa G ---> O" ),
        "mossa G ---> O",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Genero lo stato figlio P */
    gsdFiglio =
        new GameStatusDescriptor( new Character( 'P' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa G ---> P" ),
        "mossa G ---> P",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
    /* Aggiungo lo stato figlio nella lista degli stati figli */
    statiFigli.insertAtBack( gsdFiglio );

    /* Genero lo stato figlio Q */
    gsdFiglio =
        new GameStatusDescriptor( new Character( 'Q' ),
        GameStatusDescriptor.NON_MARCATO,
        valutaCostoMossa( "mossa G ---> Q" ),
        "mossa G ---> Q",
        alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
```

```
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }

    /* Se lo stato corrente è L */
    if ( currentGSD.getGameStatus().equals( new Character( 'L' ) ) ) {
        /* Allora le mosse possibili sono quelle che porteranno
         * agli stati figli R, S e T */

        /* Genero lo stato figlio R */
        GameStatusDescriptor gsdFiglio =
            new GameStatusDescriptor( new Character( 'R' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa L ---> R" ),
            "mossa L ---> R",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio S */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'S' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa L ---> S" ),
            "mossa L ---> S",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Genero lo stato figlio T */
        gsdFiglio =
            new GameStatusDescriptor( new Character( 'T' ),
            GameStatusDescriptor.NON_MARCATO,
            valutaCostoMossa( "mossa L ---> T" ),
            "mossa L ---> T",
            alternaGiocatoreDiMano( currentGSD.getHasToMove() ) );
        /* Aggiungo lo stato figlio nella lista degli stati figli */
        statiFigli.insertAtBack( gsdFiglio );

        /* Ritorno la lista dei figli */
        return statiFigli;
    }
```

C.2.2 Metodo valutaCostoMossa

```
public Float valutaCostoMossa( String moveToDone ) {
    /* Per semplicità non si entra nel merito del meccanismo di calcolo
     * dei punteggi delle mosse.
     * Semplicemente si associa un punteggio alla mossa da effettuare. */

    if ( moveToDone.equals( "mossa D ---> H" ) ) return new Float( 28 );
    if ( moveToDone.equals( "mossa D ---> I" ) ) return new Float( 31 );

    if ( moveToDone.equals( "mossa E ---> J" ) ) return new Float( 12 );
    if ( moveToDone.equals( "mossa E ---> K" ) ) return new Float( 33 );

    if ( moveToDone.equals( "mossa L ---> R" ) ) return new Float( 5 );
```

```
        if ( moveToDone.equals( "mossa L ---> S" ) ) return new Float( 3 );
        if ( moveToDone.equals( "mossa L ---> T" ) ) return new Float( 2 );

        if ( moveToDone.equals( "mossa F ---> M" ) ) return new Float( 1 );
        if ( moveToDone.equals( "mossa F ---> N" ) ) return new Float( 23 );

        if ( moveToDone.equals( "mossa G ---> O" ) ) return new Float( 22 );
        if ( moveToDone.equals( "mossa G ---> P" ) ) return new Float( 5 );
        if ( moveToDone.equals( "mossa G ---> Q" ) ) return new Float( 16 );

        return new Float( 0 );
    }
```

Appendice D

Risultati delle verifiche

D.1 Esame 25/06/2003

```
Tagliato il ramo che porta allo stato:
O
NON_MARCATO
-6.0
mossa F ---> O
MAX

Tagliato il ramo che porta allo stato:
H
NON_MARCATO
0.0
mossa C ---> H
MIN

Tagliato il ramo che porta allo stato:
J
NON_MARCATO
0.0
mossa D ---> J
MIN

lista di visita in pre-order:
A
ASSEGNATO_VALORE
1.0
nessuna mossa fatta per raggiungere lo stato iniziale
MIN

B
ASSEGNATO_VALORE
1.0
mossa A ---> B
MAX

E
ASSEGNATO_VALORE
1.0
mossa B ---> E
MIN
```

```
K
ASSEGNATO_VALORE
1.0
mossa E ---> K
MAX

L
ASSEGNATO_VALORE
5.0
mossa E ---> L
MAX

M
ASSEGNATO_VALORE
3.0
mossa E ---> M
MAX

F
ASSEGNATO_VALORE
-2.0
mossa B ---> F
MIN

N
ASSEGNATO_VALORE
-2.0
mossa F ---> N
MAX

C
ASSEGNATO_VALORE
7.0
mossa A ---> C
MAX

G
ASSEGNATO_VALORE
7.0
mossa C ---> G
MIN

P
ASSEGNATO_VALORE
7.0
mossa G ---> P
MAX

Q
ASSEGNATO_VALORE
8.0
mossa G ---> Q
MAX

D
ASSEGNATO_VALORE
1.0
mossa A ---> D
MAX

I
```

```
ASSEGNATO_VALORE
1.0
mossa D ---> I
MIN

T
ASSEGNATO_VALORE
1.0
mossa I ---> T
MAX

U
ASSEGNATO_VALORE
4.0
mossa I ---> U
MAX

V
ASSEGNATO_VALORE
2.0
mossa I ---> V
MAX
```

D.2 Esame del 18/03/2003

```
Tagliato il ramo che porta allo stato:
L
NON_MARCATO
0.0
mossa E ---> L
MIN

Tagliato il ramo che porta allo stato:
G
NON_MARCATO
0.0
mossa C ---> G
MAX

lista di visita in pre-order:
A
ASSEGNATO_VALORE
31.0
nessuna mossa fatta per raggiungere lo stato iniziale
MAX

B
ASSEGNATO_VALORE
31.0
mossa A ---> B
MIN

D
ASSEGNATO_VALORE
31.0
mossa B ---> D
MAX

H
```

```
ASSEGNATO_VALORE
28.0
mossa D ---> H
MIN

I
ASSEGNATO_VALORE
31.0
mossa D ---> I
MIN

E
ASSEGNATO_VALORE
33.0
mossa B ---> E
MAX

J
ASSEGNATO_VALORE
12.0
mossa E ---> J
MIN

K
ASSEGNATO_VALORE
33.0
mossa E ---> K
MIN

C
ASSEGNATO_VALORE
23.0
mossa A ---> C
MIN

F
ASSEGNATO_VALORE
23.0
mossa C ---> F
MAX

M
ASSEGNATO_VALORE
1.0
mossa F ---> M
MIN

N
ASSEGNATO_VALORE
23.0
mossa F ---> N
MIN
```

D.3 Esame del 18/03/2003 - variante

```
Tagliato il ramo che porta allo stato:
K
NON_MARCATO
33.0
```

```
mossa E ---> K
MAX

lista di visita in pre-order:
A
ASSEGNATO_VALORE
5.0
nessuna mossa fatta per raggiungere lo stato iniziale
MIN

B
ASSEGNATO_VALORE
28.0
mossa A ---> B
MAX

D
ASSEGNATO_VALORE
28.0
mossa B ---> D
MIN

H
ASSEGNATO_VALORE
28.0
mossa D ---> H
MAX

I
ASSEGNATO_VALORE
31.0
mossa D ---> I
MAX

E
ASSEGNATO_VALORE
12.0
mossa B ---> E
MIN

J
ASSEGNATO_VALORE
12.0
mossa E ---> J
MAX

C
ASSEGNATO_VALORE
5.0
mossa A ---> C
MAX

F
ASSEGNATO_VALORE
1.0
mossa C ---> F
MIN

M
ASSEGNATO_VALORE
1.0
mossa F ---> M
```

```
MAX

N
ASSEGNATO_VALORE
23.0
mossa F ---> N
MAX

G
ASSEGNATO_VALORE
5.0
mossa C ---> G
MIN

O
ASSEGNATO_VALORE
22.0
mossa G ---> O
MAX

P
ASSEGNATO_VALORE
5.0
mossa G ---> P
MAX

Q
ASSEGNATO_VALORE
16.0
mossa G ---> Q
MAX
```

Appendice E

Classi delle strutture dati

E.1 Classe ListNode

```
/*
 * ListNode
 * Copyright (C) 2004 Tarin Gamberini
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this program; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */


/*
 * ListNode
 * Copyright (C) 2004 Tarin Gamberini
 *
 * Questo programma è software libero; è lecito redistribuirlo o
 * modificarlo secondo i termini della Licenza Pubblica Generica
 * GNU come è pubblicata dalla Free Software Foundation; o la
 * versione 2 della licenza o (a propria scelta) una versione
 * successiva.
 *
 * Questo programma è distribuito nella speranza che sia utile, ma
 * SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di
 * NEGOZIABILITÀ o di APPLICABILITÀ PER UN PARTICOLARE SCOPO. Si
 * veda la Licenza Pubblica Generica GNU per avere maggiori
 * dettagli.
 *
 * Questo programma deve essere distribuito assieme ad una copia
 * della Licenza Pubblica Generica GNU; in caso contrario, se ne
```

```
 * può ottenere una scrivendo alla Free Software Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 */


public class ListNode {

    /* Il contenuto informativo del nodo della lista è un riferimento
     ad un oggetto generico */
    private Object data;
    /* Puntatore al successivo nodo della lista */
    private ListNode next;

    /** Crea un nuovo ListNode vuoto e senza un nodo successore */
    public ListNode() {
        /* Invoca il costruttore ListNode( Object o, ListNode nextNode ) */
        this( null , null );
    }

    /** Crea un nuovo ListNode pieno e senza successore */
    public ListNode( Object o ) {
        /* Invocaz il costruttore ListNode( Object o, ListNode nextNode ) */
        this( o , null );
    }

    /** Crea un nuovo ListNode vuoto ma con successore */
    public ListNode( ListNode nextNode ) {
        /* Invocaz il costruttore ListNode( Object o, ListNode nextNode ) */
        this( null , nextNode );
    }

    /** Crea un nuovo ListNode pieno e con successore */
    public ListNode( Object o, ListNode nextNode ) {
        data = o;
        next = nextNode;
    }

    /** Ritorna un riferimento all'oggetto contenuto nel nodo */
    public Object getObject() {
        return data ;
    }

    /** Setta l'oggetto del nodo */
    public void setObject( Object o ) {
        data = o;
    }

    /** Ritorna il puntatore al nodo successivo */
    public ListNode getNext() {
        return next;
    }

    /** Setta il puntatore al nodo successivo */
    public void setNext( ListNode nextNode ) {
        next = nextNode;
    }

}
```

E.2 Classe List

```
/*
 * List
 * Copyright (C) 2004 Tarin Gamberini
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this program; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */


/*
 * List
 * Copyright (C) 2004 Tarin Gamberini
 *
 * Questo programma è software libero; è lecito redistribuirlo o
 * modificarlo secondo i termini della Licenza Pubblica Generica
 * GNU come è pubblicata dalla Free Software Foundation; o la
 * versione 2 della licenza o (a propria scelta) una versione
 * successiva.
 *
 * Questo programma è distribuito nella speranza che sia utile, ma
 * SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di
 * NEGOZIABILITÀ o di APPLICABILITÀ PER UN PARTICOLARE SCOPO. Si
 * veda la Licenza Pubblica Generica GNU per avere maggiori
 * dettagli.
 *
 * Questo programma deve essere distribuito assieme ad una copia
 * della Licenza Pubblica Generica GNU; in caso contrario, se ne
 * può ottenere una scrivendo alla Free Software Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 */


public class List {

    private String name; /* Nome della lista */
    private int length; /* Lunghezza della lista */
    private ListNode firstNode; /* Puntatore al primo nodo della lista */
    private ListNode lastNode; /* Puntatore all'ultimo nodo della lista */

    /** Crea una nuova List */
    public List() {
        /* Invoca il costruttore List( String s ) */
        this( "list" );
    }

    /** Crea una nuova List */
    public List( Object o ) {
```

```
    /* Invoca il costruttore List( String s, Object o ) */
    this( "list", o );
}

/** Crea una nuova List in uno stato consistente */
public List( String s ) {
    name = s;
    length = 0;
    firstNode = lastNode = null;
}

/** Crea una nuova List in uno stato consistente */
public List( String s, Object o ) {
    name = s;
    length = 0;
    firstNode = lastNode = new ListNode( o );
}

/** Ritorna una String che è il nome della lista */
public synchronized String getName() {
    return name;
}

/** Ritorna un int che è rappresenta il numero di nodi presenti nella
 * lista, cioè la lunghezza della lista */
public synchronized int getLength() {
    return length;
}

/** Ritorna un riferimento al primo ListNode della lista. */
public synchronized ListNode getFirst() {
    return firstNode;
}

/** Ritorna un riferimento all'ultimo ListNode della lista. */
public synchronized ListNode getLast() {
    return lastNode;
}

/** Setta il nome della lista. */
public synchronized void setName( String lstName ) {
    name = lstName;
}

/** Ritorna true se la lista e' vuota */
public synchronized boolean isEmpty() {
    return firstNode == null;
}

/** Inserisce un Object in testa alla List */
public synchronized void insertAtFront( Object o ) {
    if ( isEmpty() )
        firstNode = lastNode = new ListNode( o );
    else
        firstNode = new ListNode( o, firstNode );
    length++;
}

/** Inserisce un Object in coda alla List */
public synchronized void insertAtBack( Object o ) {
    if ( isEmpty() )
        firstNode = lastNode = new ListNode( o );
```

```
        else {
            lastNode.setNext( new ListNode( o ) );
            lastNode = lastNode.getNext();
        }
        length++;
    }

    /** Inserisce un Object nella posizione position della lista.
     * Il nodo che si trovava nella posizione (i)esima prima dell'inserimento
     * si troverà nella posizione (i+1)esima dopo l'inserimento.
     * Il primo oggetto della lista ha posizione 1, il secondo 2 e così via. */
    public synchronized void insertAtPosition( Object o, int position )
            throws OutOfListException {
        int countPosition;

        if ( position < 1 || position > getLength() )
            throw new OutOfListException( name, position );

        if ( position == 1 )
            insertAtFront( o );
        else {
            ListNode i = getFirst();
            for ( countPosition = 1; countPosition < ( position - 1 );
                    countPosition++ )
                i = i.getNext();

            /** Salvo in tmp il riferimento al nodo che prima dell'inserimento si
             * trova in posizione position */
            ListNode tmp = i.getNext();
            /** Il nodo in posizione position-1 punta al nuovo nodo, che ora
             * viene inserito in posizione position.
             * Il nodo che prima dell'inserimento si trovava in posizione
             * position ora si trova in posizione position+1 */
            i.setNext( new ListNode( o, tmp ) );
            length++;
        }
    }

    /** Appende una lista in coda a questa. */
    public synchronized void append( List lstToAppend ) {
        if ( isEmpty() ) {
            length = lstToAppend.getLength();
            firstNode = lstToAppend.getFirst();
            lastNode = lstToAppend.getLast();
        }
        else {
            length = length + lstToAppend.getLength();
            lastNode.setNext( lstToAppend.getFirst() );
            lastNode = lstToAppend.getLast();
        }
    }

    /** Rimuove il primo nodo in testa alla List.
     * Lancia una EmptyListException( nome_della_lista ) se si tenta di
     * rimuovere un nodo da una lista vuota. */
    public synchronized Object removeFromFront() throws EmptyListException {
        Object removedNode;

        if ( isEmpty() )
            throw new EmptyListException( name );

        /* Recupera l'oggetto rimosso */
```

```
    removedNode = firstNode.getObject();

    if ( firstNode.equals( lastNode ) )
        firstNode = lastNode = null;
    else
        firstNode = firstNode.getNext();
    length--;
    return removedNode;
}

/** Rimuove l'ultimo nodo in coda alla List.
 * Lancia una EmptyListException( nome_della_lista ) se si tenta di
 * rimuovere un nodo da una lista vuota. */
public synchronized Object removeFromBack() throws EmptyListException {
    Object removedNode;

    if ( isEmpty() )
        throw new EmptyListException( name );

    /* Recupera l'oggetto rimosso */
    removedNode = lastNode.getObject();

    if ( firstNode.equals( lastNode ) )
        firstNode = lastNode = null;
    else {
        ListNode i = firstNode;
        while( i.getNext() != lastNode )
            i = i.getNext();
        lastNode = i;
        lastNode.setNext( null );
    }
    length--;
    return removedNode;
}

/** Rimuove un Object dalla posizione position della lista.
 * Il nodo che si trovava in posizione (i+1)esima prima della cancellazione,
 * si troverà in posizione (i)esima dopo la cancellazione.
 * Il primo oggetto della lista ha posizione 1, il secondo 2 e così via. */
public synchronized Object removeFromPosition( int position )
        throws OutOfListException, EmptyListException {
    int countPosition;
    Object removedNode;

    if ( isEmpty() )
        throw new EmptyListException( name );

    if ( position < 1 || position > getLength() )
        throw new OutOfListException( name, position );

    if ( position == 1 )
        return removeFromFront();
    else
        if ( position == getLength() )
            return removeFromBack();
        else
        {
            ListNode i = getFirst();
            for ( countPosition = 1; countPosition < ( position - 1 );
                    countPosition++ )
                i = i.getNext();
```

```
                /** Recupera l'oggetto rimosso */
                removedNode = i.getNext().getObject();

                /** Il nodo in posizione position-1 punta al nodo, che
                 * prima della cancellazione si trovava in posizione
                 * position+1. Il nodo che prima della cancellazione
                 * si trovava in posizione position+1 ora si trova in
                 * posizione position. */
                i.setNext( i.getNext().getNext() );
                length--;
                return removedNode;
            }
    }

    /** Ritorna la sottolista il cui primo elemento si trova in posizione
     * position nella lista chiamante il metodo. La lista chiamante avrà
     * come ultimo elemento quello in posizione position-1.
     * Il primo oggetto della lista ha posizione 1, il secondo 2 e così via.
     * per esempio data la list: 1 2 3 4 5 6 7 8 9 ed invocato
     * subList = cutFromPosition(4) avremo:
     * list: 1 2 3 mentre sublist: 4 5 6 7 8 9. */
    public synchronized List cutFromPosition( int position )
            throws OutOfListException, EmptyListException {
        int countPosition;

        if ( isEmpty() )
            throw new EmptyListException( name );

        if ( position < 1 || position > getLength() )
            throw new OutOfListException( name, position );

        /* Creo la lista tagliata */
            List cuttedList = new List( "cutted " + name );

        if ( position == 1 ) {
            /* Gli elementi della lista da tagliare vengono appesi a quelli
             * della lista tagliata */
            cuttedList.append( this );
            /* La lista chiamante il metodo cut rimane vuota */
            length = 0;
            firstNode = lastNode = null;
        }
        else {
            ListNode i = getFirst();
            for ( countPosition = 1; countPosition < ( position - 1 );
                    countPosition++ )
                 i = i.getNext();

            /* Recupera il puntatore alla catena di
             * nodi tagliati */
            ListNode j = i.getNext();
            /* L'elemento di posizione i=position-1
             * diventa l'ultimo della lista */
            i.setNext( null );
            /* Inserisco i nodi tagliati alla nuova cuttedList */
            while ( j != null ) {
                cuttedList.insertAtBack( j.getObject() );
                /* Per ogni elemento tagliato via dalla lista
                 * che invoca il metodo cut occorre diminuire
                 * la lunghezza della lista chiamante */
                length--;
                j = j.getNext();
```

```
            }
        }
        return cuttedList;
    }

    /** Invia il contenuto di List sul System.out */
    public synchronized void print() {
        if ( isEmpty() )
            System.out.println( name + " vuota" );
        else {
            System.out.print( name + ": " );

            ListNode i = firstNode;
            while( i != null ) {
                System.out.print( i.getObject().toString() + " " );
                i = i.getNext();
            }
        }
    }

}
```

E.3 Classe GeneralTreeNode

```
/*
 * GeneralTreeNode
 * Copyright (C) 2004 Tarin Gamberini
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this program; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */


/*
 * GeneralTreeNode
 * Copyright (C) 2004 Tarin Gamberini
 *
 * Questo programma è software libero; è lecito redistribuirlo o
 * modificarlo secondo i termini della Licenza Pubblica Generica
 * GNU come è pubblicata dalla Free Software Foundation; o la
 * versione 2 della licenza o (a propria scelta) una versione
 * successiva.
 *
 * Questo programma è distribuito nella speranza che sia utile, ma
 * SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di
 * NEGOZIABILITÀ o di APPLICABILITÀ PER UN PARTICOLARE SCOPO. Si
 * veda la Licenza Pubblica Generica GNU per avere maggiori
```

```
 * dettagli.
 *
 * Questo programma deve essere distribuito assieme ad una copia
 * della Licenza Pubblica Generica GNU; in caso contrario, se ne
 * può ottenere una scrivendo alla Free Software Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 */


public class GeneralTreeNode {

    private Object data; // dato da memorizzare nel nodo
    private List nexts; // Lista dei prossimi nodi: i figli

    /** Creates new GeneralTreeNode: un nodo foglia vuoto */
    public GeneralTreeNode() {
        /* Invoca il costruttore GeneralTreeNode( Object obj, List lst ) */
        this( null , null );
    }

    /** Creates new GeneralTreeNode: un nodo foglia pieno */
    public GeneralTreeNode( Object obj ) {
        /* Invoca il costruttore GeneralTreeNode( Object obj, List lst ) */
        this( obj, null );
    }

    /** Creates new GeneralTreeNode: un nodo con figli ma vuoto */
    public GeneralTreeNode( List lst ) {
        this( null , lst );
    }

    /** Creates new GeneralTreeNode: un nodo con figli pieno */
    public GeneralTreeNode( Object obj, List lst ) {
        data = obj;
        if ( lst == null )
            nexts = new List();
        else
            nexts = lst;
    }

    /** Ritorna un riferimento all'oggetto contenuto nel nodo */
    public Object getObject() {
        return data ;
    }

    /** Setta l'oggetto del nodo */
    public void setObject( Object obj ) {
        data = obj;
    }

    /** Ritorna il riferimento di testa alla lista dei figli */
    public List getNexts() {
        return nexts;
    }

    /** Setta il riferimento di testa alla lista dei figli */
    public void setNexts( List lst ) {
        nexts = lst;
    }

    /** Ritorna vero se il nodo è una foglia, ossia il riferimento
     * alla lista dei figli è null */
```

```
    public boolean isLeaf() {
        return ( nexts == null );
    }

}
```

E.4 Classe GeneralTree

```
/*
 * GeneralTree
 * Copyright (C) 2004 Tarin Gamberini
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
 * License along with this program; if not, write to the Free
 * Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */


/*
 * GeneralTree
 * Copyright (C) 2004 Tarin Gamberini
 *
 * Questo programma è software libero; è lecito redistribuirlo o
 * modificarlo secondo i termini della Licenza Pubblica Generica
 * GNU come è pubblicata dalla Free Software Foundation; o la
 * versione 2 della licenza o (a propria scelta) una versione
 * successiva.
 *
 * Questo programma è distribuito nella speranza che sia utile, ma
 * SENZA ALCUNA GARANZIA; senza neppure la garanzia implicita di
 * NEGOZIABILITÀ o di APPLICABILITÀ PER UN PARTICOLARE SCOPO. Si
 * veda la Licenza Pubblica Generica GNU per avere maggiori
 * dettagli.
 *
 * Questo programma deve essere distribuito assieme ad una copia
 * della Licenza Pubblica Generica GNU; in caso contrario, se ne
 * può ottenere una scrivendo alla Free Software Foundation,
 * Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
 */


public class GeneralTree {

    /* Nome dell'albero */
    private String name;
    /* Numero di nodi */
    private int numberOfNodes;
    /* Radice dell'albero */
```

```
    private GeneralTreeNode root;

    /** Creates new GeneralTree vuoto.*/
    public GeneralTree() {
        this( "tree" );
    }

   /** Creates new GeneralTree vuoto e con nome treeName. */
    public GeneralTree( String treeName ) {
        name = treeName;
        numberOfNodes = 0;
        root = null;
    }

    /** Creates new GeneralTree formato da una sola foglia. */
    public GeneralTree( Object objToInsert ) {
        this( "tree", objToInsert );
    }

    /** Creates new GeneralTree formato da una sola foglia e
     * con nome treeName. */
    public GeneralTree( String treeName, Object objToInsert ) {
        name = treeName;
        numberOfNodes = 1;
        root = new GeneralTreeNode( objToInsert );
    }

    /** Ritorna il nome dell'albero.*/
    public String getName() {
        return name;
    }

    /** Ritorna il numero di nodi dell'albero.*/
    public int getNumberOfNodes() {
        return numberOfNodes;
    }

    /** Ritorna la radice dell'albero.*/
    public GeneralTreeNode getRoot() {
        return root;
    }

    /** Ritorna true se l'albero è vuoto, ossia se la radice
     * dell'albero vale null */
    public boolean isEmpty() {
        return root == null;
    }

    /** Genera il sottoalbero figlio del nodo <CODE>fatherNode</CODE>,
     * assegnando al figlio il contenuto <CODE>objSon</CODE>. Il figlio è
     * inserito alla fine della lista dei figli di fatherNode, in altre
     * parole il figlio è l'ultimo dei fratelli. */
    public void insertAtBackNodeSon( GeneralTreeNode fatherNode,
            Object objSon ) {
        /* Aggiungo un riferimento al figlio nella lista dei
         * figli presente nel padre */
        fatherNode.getNexts().insertAtBack( new GeneralTree( objSon ) );
        numberOfNodes++;
    }

    /** Dato l'albero father ed una lista lstOfObjectToInsertAsSons
     * di oggetti da inserire come figli, il metodo crea un sottoalbero
```

```
 * per ogni oggetto della lista e lo collega al nodo padre consecutivamente
 * agli eventuali figli di father già presenti. */
public void appendListOfSons( GeneralTree father,
        List lstOfObjectToInsertAsSons ) throws EmptyGeneralTreeException {

    /* Se la radice dell'albero punta a null non ha senso appendere figli
     * perchè non esiste un padre per loro. */
    if ( father.isEmpty() )
        throw new EmptyGeneralTreeException( father.getName() );

    /* Se la lista di figli non è vuota */
    if ( !lstOfObjectToInsertAsSons.isEmpty() ) {
        /* si procede con l'adozione dei figli da parte del padre */

        /* Puntatore alla lista degli oggetti da inserire come figli */
        ListNode pObjectList = lstOfObjectToInsertAsSons.getFirst();
        while ( pObjectList != null ) {
            /* Creo un sottoalbero figlio */
            GeneralTree sonTree = new GeneralTree( pObjectList.getObject() );
            /* Lo collego (adottato) al padre*/
            father.getRoot().getNexts().insertAtBack( sonTree );
            /* Proseguo col figlio successivo*/
            pObjectList = pObjectList.getNext();
        }
    }
    /* Se la lista dei figli è vuota non accade nulla in quanto il padre
     * non ha alcun figlio da adottare */
}

/** Visita in pre-oreder: prima il nodo padre, poi i sottoalberi figli */
public List preOrder( GeneralTree treeToVisit )
        throws EmptyGeneralTreeException {
    /* Crea una lista vuota come risultato della visita in pre-order */
    List preOrderedList = new List( "lista di visita in pre-order" );

    if ( treeToVisit.isEmpty() )
        throw new EmptyGeneralTreeException( treeToVisit.getName() );

    /* Visito la radice */
    preOrderedList.insertAtFront( treeToVisit.getRoot().getObject() );
    /* Se ci sono sottoalberi figli, ossia se la lista dei sottoalberi
     * figli non è vuota */
    if ( !treeToVisit.getRoot().getNexts().isEmpty() ) {
        /* Visito i figli attraverso il puntatore pListNode che scorre
         * lungo la lista dei figli. */
        ListNode pListNode = treeToVisit.getRoot().getNexts().getFirst();
        while ( pListNode != null ) {
            /* L'oggetto contenuto come dato nella lista dei figli è un
             * riferimento ad un sottoalbero figlio dell'albero.
             * Visito il sottoalbero figlio pListNode.getObject() */
            preOrderedList.append(
                    preOrder( (GeneralTree)pListNode.getObject() ) );
            /* Proseguo col figlio successivo */
            pListNode = pListNode.getNext();
        }
    }
    /* Ritorna la lista di visita in pre-order */
    return preOrderedList;
}

/** Visita in post-oreder: prima i sottoalberi figli, poi il nodo padre */
public List postOrder( GeneralTree treeToVisit )
```

```
            throws EmptyGeneralTreeException {
        /* Crea una lista vuota come risultato della visita in post-order */
        List postOrderedList = new List( "lista di visita in post-order" );

        if ( treeToVisit.isEmpty() )
            throw new EmptyGeneralTreeException( treeToVisit.getName() );

        /* Se ci sono sottoalberi figli, ossia se la lista dei sottoalberi
         * figli non è vuota */
        if ( !treeToVisit.getRoot().getNexts().isEmpty() ) {
            /* Visito i figli attraverso il puntatore pListNode che scorre
             * lungo la lista dei figli. */
            ListNode pListNode = treeToVisit.getRoot().getNexts().getFirst();
            while ( pListNode != null ) {
                /* L'oggetto contenuto come dato nella lista dei figli è un
                 * riferimento ad un sottoalbero figlio dell'albero.
                 * Visito il sottoalbero figlio pListNode.getObject() */
                postOrderedList.append(
                        preOrder( (GeneralTree)pListNode.getObject() ) );
                /* Proseguo col figlio successivo */
                pListNode = pListNode.getNext();
            }
        }
        /* Visito la radice */
        postOrderedList.insertAtBack( treeToVisit.getRoot().getObject() );
        /* Ritorna la lista di visita in pre-order */
        return postOrderedList;
    }

}
```

Appendice F

GNU Free Documentation License

Version 1.2, November 2002
Copyright ©2000,2001,2002 Free Software Foundation, Inc.

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as

"you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To

“Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.