Bantumi 4

Questo esercizio è facoltativo, ma chi decide di farlo **deve consegnarlo entro il 1/7/2015**. Alla consegna il vostro programma viene fatto giocare contro il mio. Se il vostro programma vince, passa la prima fase. Nella seconda fase i programmi che hanno passato la prima fase vengono fatti giocare tra loro. Verrà stilata una classifica dei migliori programmi, in funzione del numero di vittorie conseguite. Questi programmi verranno anche visionati da me e la loro chiarezza e semplicità avrà un peso sul giudizio finale. Ai migliori programmi verranno attribuiti al massimo 3 punti. Chi passa le eliminatorie avrà al massimo 2 punti. Chi consegna un programma che vince qualche partita contro il mio avrà al massimo 1 punto. I punti conseguiti verranno aggiunti a quelli dell'esame (che deve essere già sufficiente)

L'esercizio richiede di realizzare una funzione che giochi a Bantumi in modo "intelligente". Questa funzione deve soddisfare la seguente specifica:

PRE=(B rappresenta la situazione del gioco, player=0/1, livello>=0)

int meglio_mossa(int B[][7],int player,int livello)

POST=(restituisce una mossa m per player, 0<=m<=5 e M[player][m]!=0)

Il vostro programma deve stare su un file che si chiama EXTRA.cpp. Dopo le solite istruzioni
#include<iostream>
#include<fstream>
using namespace std;

il vostro programma deve contenere la dichiarazione :

namespace EXTRA{

//qui va inserito tutto quello che serve alla vostra funzione meglio_mossa

}//questa è l'ultima riga del programma

Il vostro programma non deve contenere un main. Quando consegnate il vostro file nell'esercizio Bantumi 4 del moodle esso verrà fatto giocare contro il mio programma. L'output mostrerà il risultato di queste partite specificando quante partite ha vinto il giocatore 0 (io) e quante ne ha vinte il giocatore 1 (voi).

Per mostrare questo output, sfruttiamo il test automatico dei programmi che in questo caso non ci serve visto che comunque l'output (cioè il risultato delle partite) non è prevedibile. Quindi stabilisco un output atteso di fantasia, per esempio, "vediamo chi vince!!" che sarà sempre diverso da quello prodotto dal programma e quindi l'output del programma vi verrà mostrato come fosse un errore. Ogni mossa prodotta dalla vostra funzione meglio_mossa viene controllata dal mio programma e in caso la mossa sia sballata (o fuori da [0..5] o indichi una buca vuota) viene prodotto un output che segnala l'errore e il programma termina.

Come già detto prima, le consegne che battono il mio programma verranno fatte giocare tra loro. Per questa seconda fase è sufficiente la consegna del Bantumi 4, non dovrete fare nulla più di questo. I risultati finali verranno pubblicati su questo moodle.

Bantumi 4 minmax

In questo documento cerco di dare qualche consiglio su come si può sviluppare una funzione che sia capace di giocare in modo intelligente un gioco. L'approccio che descriverò è infatti generale e si applica a qualsiasi gioco in cui ci siano 2 avversari che si affrontano. La tecnica si chiama minmax. E' facile trovare in rete informazioni su minmax, per esempio in http://web.stanford.edu/~msirota/soco/minimax.html.

Questa tecnica si basa su 2 punti:

a) la capacità di valutare le prospettive di vittoria di una certa configurazione di gioco per il giocatore che deve muovere da quella configurazione. Questa valutazione viene fatta senza giocare mosse in avanti nella partita, ma solamente sulla base di un esame della configurazione. Chiameremo questo esame **analisi statica** della configurazione, per sottolineare il fatto che non comporta l'effettuazione di mosse di gioco in avanti.

b) la capacità di giocare tutte le possibili mosse passando alternativamente da un giocatore all'altro in modo da generare tutte le configurazioni di gioco raggiungibili da quella iniziale in un certo numero di mosse che chiameremo n_mosse. Con n_mosse grande, sarebbe possibile, in teoria, generare tutte le partite raggiungibili dalla configurazione iniziale. In questo modo, avremmo completa conoscenza del gioco (il che non garantisce necessariamente la vittoria, infatti questo dipende dalla configurazione iniziale). Visto che nel gioco del Bantumi, a partire da una qualsiasi configurazione ci sono (in generale) 6 mosse possibili, il numero di configurazioni raggiungibili in n_mosse mosse è circa 6^{n_mosse}. E' facile capire che se n_mosse cresce il numero di configurazioni raggiunte diventa rapidamente ingestibile. Quindi noi considereremo n_mosse=3 o 4. Lo sviluppo delle configurazioni raggiungibili con n_mosse mosse è tipicamente rappresentato da un albero che chiameremo **l'albero delle mosse**. Nel link dato sopra trovate un esempio di un tale albero.

Complessivamente l'idea del minmax è la seguente:

i) si genera l'albero delle mosse, alla sua frontiera avremo tutte le configurazioni raggiungibili in n_mosse mosse (per un fissato valore di n_mosse);

ii) a ciascuna di queste configurazioni viene attribuito un voto attraverso la funzione di analisi statica;

iii) questi voti vengono fatti risalire dalle foglie verso la radice dell'albero in questo modo che ogni possibile mossa a partire dalla configurazione iniziale riceva un voto. Ovviamente viene scelta la mossa col voto migliore.

Il procedimento (iii) di risalita dei voti dalle foglie alla radice dell'albero delle mosse è il cuore del metodo e spiega anche il nome minmax di questa tecnica. Infatti in generale dopo ogni mossa il giocatore cambia e quindi anche il trattamento dei voti cambia. Se il giocatore 0 deve giocare, farà tutte le sue mosse e per ciascuna di esse il giocatore 1 proverà tutte le sue. Fermiamoci qui, cioè fissiamo n_mosse=2. Dalle configurazioni alla frontiera è il giocatore 0 che deve giocare. Quindi con l'analisi statica queste configurazioni ricevono un voto che rispecchia le chance di successo di 0 per ciascuna di esse. Il giocatore 1 dovrà scegliere la sua mossa che porta al **minimo** di chance per il giocatore 0. Questa scelta attribuisce un voto a ciascuna delle configurazioni da cui il giocatore 1 muove. Al livello superiore, il giocatore 0 vede questi voti associati alle configurazioni che può raggiungere con 1 mossa e sceglie la mossa che lo porta alla configurazione con il voto **massimo**. Infatti questa mossa porta ad una configurazione a partire dalla quale il giocatore 1, nonostante provi a minimizzare le possibilità di 0, ci riesce meno che negli altri casi.

Quindi il giocatore 1 sceglie il **minimo** dei voti delle configurazioni che può raggiungere con una mossa mentre il giocatore 0 sceglie il **massimo** dei voti delle configurazioni che raggiunge con una mossa. Se questa spiegazione non vi convince pienamente, all'indirizzo già indicato, http://web.stanford.edu/~msirota/soco/minimax.html trovate una figura ed una spiegazione chiara.

Dal punto di vista della realizzazione del progetto, è importante capire che non è affatto necessario costruire l'albero di altezza n_mosse delle configurazioni raggiungibili in n_mosse. La ricorsione ci permette di generare tutte le configurazioni di quest'albero attraverso un'opportuna sequenza di invocazioni ricorsive. La funzione ricorsiva che si occupa di calcolare la miglior mossa per il giocatore 0, eseguirà 6 invocazioni ricorsive che corrispondono alle 6 mosse possibili e ricorsivamente l'invocazione che gestisce una di queste 6 configurazioni farà altre 6 invocazioni ricorsive che corrisponderanno alle mosse del giocatore 1 e così via. In questo modo le chiamate ricorsive percorrono l'albero delle mosse in profondità (depth-first) e da sinistra a destra.

Inoltre osserviamo che nel gioco del Bantumi, non sempre il giocatore cambia dopo ogni mossa e quindi l'albero delle mosse è meno regolare di quello di http://web.stanford.edu/~msirota/soco/minimax.html. In presenza di più mosse consecutive di uno stesso giocatore, suggeriamo di non decrementare n_mosse che indicherà quindi non tanto il numero di mosse in avanti che vogliamo simulare, ma il numero di cambi di giocatore da considerare. Se si segue quanto suggerito, se n_mosse è pari allora alla frontiera dell'albero delle mosse avremo sempre configurazioni in cui deve giocare lo stesso giocatore che gioca nella configurazione iniziale, mentre se n_mosse è dispari alla frontiera ci saranno configurazioni da cui muove il giocatore diverso da quello iniziale. In entrambi i casi la procedura minmax è la stessa.

Da quanto detto dovrebbe essere comprensibile che minmax produce una condotta di gioco prudente in cui il giocatore cerca di impedire all'avversario di vincere, anziché provare a vincere lui.

Nel mio programma n_mosse è chiamato livello perché è in relazione al numero di livelli dell'albero delle mosse. Livello 3 e 4 è menzionato nell'output del programma.