

# Il problema della rana affamata

Progetto di Laboratorio di Algoritmi e Strutture Dati  
Anno AA 2006/07

Docente: Ezio Bartocci

web site: <https://unicam.it/~ezio.bartocci>

Dipartimento di Matematica e Informatica

Università di Camerino

1 Gennaio 2007

## 1 Introduzione

Una rana viene paracadutata su un campo di cibo per anfibi. Il campo è un quadrato costituito da  $n*n$  caselle disposte in una matrice di  $n$  righe e  $n$  colonne. Ogni casella contiene un'unità di cibo. La rana effettua salti di lunghezza prefissata e in otto direzioni diverse. Così come mostrato in Fig. 1, se dovesse trovarsi, ad esempio, nella casella di riga 4 e colonna 4 (4,4), muovendosi in orizzontale potrebbe raggiungere le due caselle (4,1) e (4,7) a seconda se si sceglie il verso sinistro o destro. Spostamenti verticali permettono di raggiungere le caselle (7,4) e (1,4). Saltando in obliquo la rana può raggiungere quattro caselle diverse che sono (2,2), (2,6), (6,6) e (6,2). Se la rana invece si trovasse in posizione (3,4) non si potrebbe muovere in direzione nord, perché uscirebbe dal campo di azione. L'esempio chiarisce i salti consentiti dalla rana che sono: due orizzontali (a est e ovest) e due verticali (a nord e sud) a distanza di tre caselle e quattro obliqui (nord-est, nord-ovest, sud-est, sud-ovest) a distanza di due caselle (in diagonale). Una volta che salta dalla prima casella alla seconda mangia il cibo e deposita un bussolotto con un numero crescente nella casella lasciata partendo dal numero 1. L'obiettivo per la rana è quello di mangiare tutto il cibo senza mai tornare su una casella senza cibo o su una non permessa, ossia fuori dal campo di azione.

## 2 Backtracking e ricorsione

La soluzione al problema si avvale della tecnica conosciuta come backtracking. Tale tecnica esplora tutte le possibili soluzioni. Poiché la soluzione può non essere unica (come potrebbe anche non esserci) in alcuni casi la prima utile che si incontra interrompe l'algoritmo. Quando si arriva a un "vicolo cieco", ovvero nella costruzione della soluzione, quando non si ha alcuna possibilità di prosecuzione, si torna indietro (da qui back) e si esplora la prima soluzione parziale (in questo viaggio a ritroso) che fornisce altre possibilità ancora non valutate. Il backtracking può essere attuato in due modi, che in fondo si riconducono ad una

stessa filosofia; con l'uso della ricorsione oppure gestendo opportunamente uno stack. La filosofia è la stessa se si pensa che il compilatore gestisce la ricorsione con uno stack. Con la ricorsione si risparmia codice. Il secondo metodo è un po' più articolato, ma permette di seguire tutti i passaggi tipici del backtracking per l'esplorazione delle sotto soluzioni.

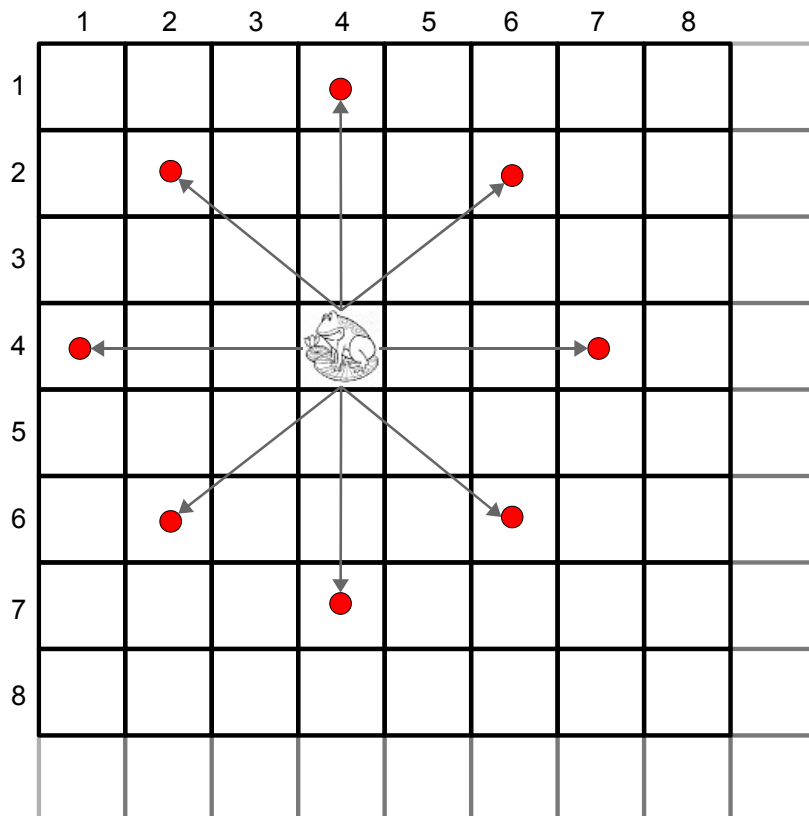


Figure 1: Mosse consentite alla rana

### 3 Scopo del progetto

Implementare la tecnica del backtracking per risolvere il problema della rana affamata utilizzando una pila o stack come struttura dati per l'esplorazione di tutte le sottosoluzioni. Nella Fig. 2 sono descritte le interfacce da implementare nel progetto. La classe che rappresenterà il campo della rana dovrà implementare i metodi definiti dall'interfaccia **Field**:

- *int maxToken()* - Restituisce il numero dei bussolotti lasciati nel campo.

- *int readToken(Coordinate xy)* - Legge il numero del bussolotto nella posizione (x,y) del campo. Restituisce 0 se quella posizione non é mai stata occupata dalla rana.
- *int size()* - restituisce la dimensione (il lato) del campo in numero di celle.

Le coordinate nel campo dovranno essere rappresentate da una classe che implementa l'interfaccia **Coordinate**:

- *int x()* - restituisce la coordinata x.
- *int y()* - restituisce la coordinata y.

Il salto compiuto dalla rana dovr  essere rappresentato da una classe che implementa l'interfaccia **JumpCoordinate**:

- *int x()* - (Metodo ereditato dall'interfaccia Coordinate) restituisce la coordinata x dopo aver effettuato un salto.
- *int y()* - (Metodo ereditato dall'interfaccia Coordinate) restituisce la coordinata y dopo aver effettuato un salto.
- *int lastDirection()* - restituisce il codice della direzione dell'ultimo salto effettuato.

La struttura dati a pila che conterr  i salti della rana dovr  essere rappresentata da una classe che implementa l'interfaccia **JumpsStack**:

- *public JumpCoordinate push(JumpCoordinate m);* - inserisce nello stack l'elemento **JumpCoordinate** e restituisce ci  che   stato inserito.
- *public JumpCoordinate pop();* - estrae dallo stack l'ultimo elemento **JumpCoordinate** che   stato inserito. Restituisce **null** se lo stack   vuoto.
- *public JumpCoordinate top();* - restituisce senza estrarlo dallo stack l'ultimo elemento **JumpCoordinate** che   stato inserito.
- *public boolean isEmpty();* - restituisce *true* se lo stack   vuoto e *false* se lo stack   pieno.
- *public int size();* - restituisce il numero di elementi presenti nello stack.

La classe che risolve il problema dovr  implementare i metodi definiti dall'Interfaccia **SolveFrogField**:

- *Field solve (int fieldSize, Coordinate start)* - risolve il problema della rana affamata in un campo di dimensione *fieldSize* e considerando *start* come posizione iniziale della rana.
- *Field partialSolve (int fieldSize, Coordinate start, int maximumToken)* - risolve parzialmente il problema della rana affamata in un campo di dimensione *fieldSize* e considerando *start* come posizione iniziale della rana e dopo aver depositato il numero *maximumToken* di bussolotti nel campo.

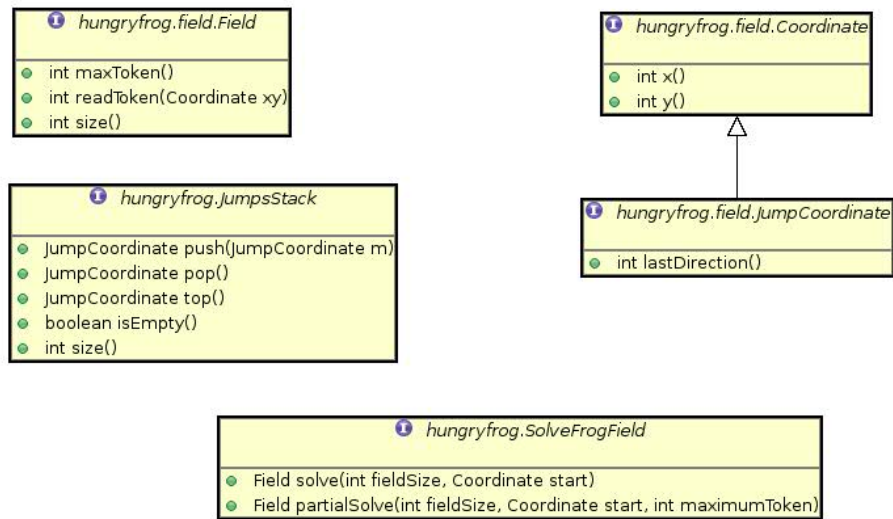


Figure 2: Interfacce Java da implementare

## 4 Modalità

Per la realizzazione di questo progetto si prega di utilizzare il compilatore fornito dalla Java SDK 1.4.2, ma non potranno essere utilizzate le Classi fonite da Java. Ulteriore materiale sarà disponibile alla pagina del corso.