

Problema del Giro del Cavallo

1 Specifica del Problema

Scrivere un programma in Haskell e uno in Prolog che acquisiscano da tastiera un naturale $N \geq 1$ che rappresenta la dimensione N della scacchiera, la quale sarà una scacchiera $N \times N$. Inoltre, il programma deve acquisire una coppia di coordinate che rappresentano la posizione di partenza del cavallo sulla scacchiera. Il programma deve risolvere il problema del giro del cavallo entro due minuti, trovando un percorso in cui il cavallo visita ogni casella della scacchiera esattamente una volta.

2 Analisi del Problema

2.1 Dati in Ingresso del Problema

I dati di ingresso del problema sono rappresentati da:

- Un numero naturale $N \geq 1$;
- Una coppia ordinata di numeri naturali (x, y) dove $x, y \in \{0, 1, \dots, N - 1\}$.

2.2 Dati in Uscita del Problema

Il dato in uscita è una matrice quadrata A di dimensione $N \times N$ contenente numeri naturali distinti.

2.3 Relazioni Intercorrenti tra i Dati

Sia N un numero naturale tale che $N \geq 1$. Una **coppia ordinata** (x, y) di numeri naturali da 0 a $N - 1$ è un elemento dell'insieme prodotto cartesiano $\{0, 1, 2, \dots, N - 1\} \times \{0, 1, 2, \dots, N - 1\}$. Formalmente, la coppia ordinata (x, y) è definita come:

$$(x, y) \in \{0, 1, 2, \dots, N - 1\} \times \{0, 1, 2, \dots, N - 1\}$$

dove:

$$x \in \{0, 1, 2, \dots, N - 1\} \quad \text{e} \quad y \in \{0, 1, 2, \dots, N - 1\}$$

La proprietà della coppia ordinata che ci interessa è la seguente:

- La coppia ordinata (x, y) è distinta dalla coppia (y, x) a meno che $x = y$.

Sia N un numero naturale tale che $N \geq 1$. Una **matrice quadrata** A di dimensione $N \times N$ è una funzione:

$$A : \{0, \dots, N - 1\} \times \{0, \dots, N - 1\} \rightarrow \{1, \dots, N^2\}$$

dove:

- $\{0, \dots, N - 1\}$ è l'insieme degli indici delle righe e delle colonne della matrice.
- $\{1, \dots, N^2\}$ è l'insieme dei valori che gli elementi della matrice possono assumere.

Formalmente, per ogni coppia di indici (i, j) con $i, j \in \{0, \dots, N - 1\}$, l'elemento della matrice A nella posizione (i, j) è denotato da $A(i, j)$ ed è un valore appartenente all'insieme $\{1, \dots, N^2\}$.

Nel contesto del problema, consideriamo il **movimento del cavallo** nel gioco degli scacchi. Il cavallo si sposta in una traslazione a forma di "L", che può essere descritta come un cambiamento nelle coordinate di una coppia ordinata (x, y) di una delle seguenti forme:

$$(x \pm 2, y \pm 1) \quad \text{oppure} \quad (x \pm 1, y \pm 2)$$

In altre parole, il cavallo può muoversi da una posizione (x, y) a una nuova posizione (x', y') se e solo se la differenza assoluta tra le coordinate soddisfa una delle seguenti condizioni:

$$\begin{cases} |x' - x| = 2 & \text{e} & |y' - y| = 1 \\ |x' - x| = 1 & \text{e} & |y' - y| = 2 \end{cases}$$

Questo movimento a forma di "L" è fondamentale per determinare la validità degli spostamenti del cavallo nella matrice A considerata.

3 Progettazione dell'Algoritmo

3.1 Scelte di Progetto

Per risolvere il problema del giro del cavallo, abbiamo utilizzato l'algoritmo di **Warnsdorff**. L'algoritmo di Warnsdorff è preferito per la sua efficienza e semplicità. Essendo un algoritmo euristico, riduce significativamente il numero di percorsi da esplorare rispetto al backtracking puro, rendendolo meno intensivo dal punto di vista computazionale e più veloce. Questo algoritmo minimizza le possibilità di stallo del cavallo scegliendo sempre la mossa che porta alla casella con il minor numero di mosse successive possibili, prevenendo blocchi e facilitando il completamento del giro.

Tuttavia, è importante notare che, a differenza dell'algoritmo di backtracking, l'algoritmo di Warnsdorff non garantisce sempre di trovare una soluzione quando esiste. In alcuni casi, può fallire, mentre l'algoritmo di backtracking, esplorando tutte le possibili combinazioni, trova sempre la soluzione quando esiste, anche se con un costo computazionale più elevato.

Abbiamo deciso di implementare la strategia di **Squirrel** che migliora l'algoritmo di Warnsdorff in quanto aggiungendo un criterio di ordinamento secondario basato sulla distanza euclidea dal centro della scacchiera, distribuisce meglio i movimenti del cavallo ed evita congestioni al centro della scacchiera nelle fasi iniziali del tour.

La dimensione in ingresso della scacchiera è stata limitata a $5 \leq N \leq 70$ poiché è dimostrato che per scacchiere di dimensione inferiore a 5×5 non esistono giri che risolvano il problema, mentre l'algoritmo di Warnsdorff-Squirrel è limitato a scacchiere di grandezza massima 70×70 .

Poiché la matrice non è una struttura dati nativa nei due linguaggi utilizzati, sia in Haskell che in Prolog, la scacchiera è rappresentata come una lista di liste, una struttura dati bidimensionale, che viene inizializzata a seguito della prima acquisizione, indicando che nessuna delle caselle è ancora stata visitata.

Le coordinate sulla scacchiera sono rappresentate come coppie di interi. La seconda acquisizione dei dati, imposta la casella dell'utente a 1, segnalando il primo passo del cavallo.

I possibili movimenti del cavallo, che negli scacchi si muove ad L, sono stati definiti come coppie di incrementi di coordinate rispetto alla posizione corrente (es. (2, 1), (1, 2), ...).

La soluzione viene stampata sotto forma di tabella contenente numeri interi che rappresentano il numero della mossa eseguito dal cavallo su ciascuna casella, facilitando la visualizzazione del percorso di quest'ultimo.

Al fine di seguire quella che è la specifica di progetto, abbiamo deciso di implementare un timeout nel caso in cui l'algoritmo euristico impieghi più di 2 minuti per trovare una soluzione.

3.2 Passi dell'Algoritmo

1. Acquisizione di un numero intero che rappresenta la dimensione della scacchiera;
2. Acquisizione di una coppia di interi che rappresentano le coordinate di partenza del cavallo;
3. Esecuzione del giro (**Algoritmo di Warnsdorff**):

- Caso Base: verifica se il numero della mossa corrente è uguale a $N \times N$. In caso affermativo, il tour è completo e la scacchiera viene restituita come soluzione.
- Caso Induttivo:
 - (a) Calcolo di tutte le mosse valide dal punto corrente. Una mossa è valida se la destinazione è all'interno dei confini della scacchiera e non è stata visitata.
 - (b) Per ciascuna mossa valida, calcolo dell'accessibilità, cioè il numero di mosse successive possibili a partire dalla destinazione della mossa.
 - (c) Ordinamento delle mosse valide in base all'accessibilità in ordine crescente.
 - (d) **Applicazione della strategia di Squirrel**: in caso di parità di accessibilità, ordinamento ulteriore delle mosse in base alla distanza dal centro della scacchiera in ordine decrescente.
 - (e) Selezione della mossa con il minor grado (e più lontana dal centro in caso di parità) e aggiornamento della scacchiera con il numero della mossa corrente.
 - (f) Si procede ricorsivamente per la nuova posizione del cavallo con la mossa successiva.

4. Terminazione:

- Al termine dell'algoritmo, se il tour è stato completato con successo, stampa della scacchiera sotto forma di tabella con i numeri delle mosse del cavallo in ordine crescente.
- Al termine del tempo stabilito, se il tour non è ancora stato completato con successo, stampa di un avviso indicando che l'esecuzione è stata interrotta prematuramente.

4 Implementazione dell'Algoritmo

File sorgente squirrelWarnsdorff.hs:

```
{- Programma Haskell per risolvere il problema del giro del cavallo usando
   l'algoritmo di Warnsdorff-Squirrel. -}

import Data.List (sortBy)
{- La libreria Data.List e' necessaria per utilizzare la funzione sortBy che ordina
   una lista in base a un criterio specifico. -}

import Data.Ord (comparing)
{- La libreria Data.Ord e' necessaria per utilizzare la funzione comparing che crea
   un criterio di ordinamento basato su una funzione di proiezione. -}

import Data.Maybe (listToMaybe)
{- La libreria Data.Maybe e' utilizzata per la funzione listToMaybe che converte una
   lista in un Maybe, prendendo il primo elemento della lista se esiste. -}

import Data.IORef (IORef, newIORef, readIORef, writeIORef)
{- La libreria Data.IORef e' utilizzata per creare e manipolare riferimenti mutabili
   (IORef) necessari per gestire lo stato mutabile nel programma. -}

import Text.Read (readMaybe)
{- La libreria Text.Read e' necessaria per utilizzare la funzione readMaybe che tenta
   di leggere un valore da una stringa e restituisce Maybe. -}

import System.Timeout (timeout)
{- La libreria System.Timeout e' necessaria per utilizzare la funzione timeout che
   termina l'esecuzione a seguito di un tempo prestabilito. -}

-- Coordinate del cavallo
type Posizione = (Int, Int)
-- Scacchiera che contiene le mosse che compie il cavallo
type Scacchiera = [[Int]]

-- Movimenti del cavallo
mosseCavallo :: [Posizione]
mosseCavallo = [(2, 1), (1, 2), (-1, 2), (-2, 1), (-2, -1), (-1, -2), (1, -2), (2, -1)]

main :: IO ()
main = do
    dimensione <- leggiDimensioneScacchiera
    posizioneIniziale <- leggiPosizione dimensione
    putStrLn "Attendere la soluzione..."
    risultato <- timeout (120 * 1000000) (risolviGiroCavallo dimensione posizioneIniziale)
    case risultato of
        Just (Just soluzione) -> stampaScacchiera soluzione
        Just Nothing -> putStrLn "Soluzione non trovata."
        Nothing -> putStrLn "Tempo scaduto. Non e' stata trovata una soluzione entro il
        tempo limite."

{- Legge la dimensione della scacchiera da tastiera e verifica la validita'.
   - restituisce un intero che rappresenta la dimensione della scacchiera. -}
leggiDimensioneScacchiera :: IO Int
```

```
leggiDimensioneScacchiera = do
  putStrLn "Inserisci la dimensione della scacchiera (intero compreso tra 5 e 70):"
  datoIngresso <- getLine
  case readMaybe datoIngresso of
    Just dimensione
      | dimensione < 5 -> do
        putStrLn "Non esiste una soluzione per scacchiere di dimensioni inferiori
a 5."
        leggiDimensioneScacchiera
      | dimensione > 70 -> do
        putStrLn "Con questo algoritmo non e' possibile risolvere il problema per
scacchiere di dimensioni superiori a 70."
        leggiDimensioneScacchiera
      | otherwise -> return dimensione
    Nothing -> do
      putStrLn "Dimensione non valida. Inserisci un numero intero."
      leggiDimensioneScacchiera

{- Legge una posizione da tastiera e verifica la validita' rispetto alla dimensione della
scacchiera.
- l'unico argomento e' la dimensione della scacchiera.
- restituisce una coppia di interi che rappresenta la posizione. -}
leggiPosizione :: Int -> IO (Int, Int)
leggiPosizione dimensione = do
  putStrLn "Inserisci la posizione di partenza del cavaliere in formato (X,Y)"
  putStrLn $ "(X e Y interi compresi tra 0 e " ++ show (dimensione - 1) ++ "):"
  datoIngresso <- getLine
  case reads datoIngresso of
    [(pos, "")] -> return pos
    _ -> do
      putStrLn "datoIngresso non valido. Inserisci di nuovo (esempio: (3,3)):"
      leggiPosizione dimensione

{- Inizializza una scacchiera NxN a -1.
- il primo argomento e' la dimensione della scacchiera. -}
inizializzaScacchiera :: Int -> Scacchiera
inizializzaScacchiera dimensione = replicate dimensione (replicate dimensione (-1))

{- Verifica se una posizione e' valida.
- il primo argomento e' la dimensione della scacchiera.
- il secondo argomento e' la scacchiera.
- il terzo argomento e' la posizione da verificare. -}
mossaValida :: Int -> Scacchiera -> Posizione -> Bool
mossaValida dimensione scacchiera (x, y) =
  x >= 0 && x < dimensione &&
  y >= 0 && y < dimensione &&
  (scacchiera !! x !! y) == -1

{- Calcola il numero di mosse valide successive da una data posizione.
- il primo argomento e' la dimensione della scacchiera.
- il secondo argomento e' la scacchiera.
- il terzo argomento e' la posizione da cui calcolare l'accessibilita'. -}
calcolaAccessibilita :: Int -> Scacchiera -> Posizione -> Int
calcolaAccessibilita dimensione scacchiera (x, y) =
  length $ filter (mossaValida dimensione scacchiera) [(x + dx, y + dy) | (dx, dy) <-
mosseCavallo]
```

```
{- Ordina le mosse in base all'accessibilita' e alla distanza dal centro.
- il primo argomento e' la dimensione della scacchiera.
- il secondo argomento e' la scacchiera.
- il terzo argomento e' la lista delle posizioni da ordinare. -}
ordinaMosse :: Int -> Scacchiera -> [Posizione] -> [(Int, Posizione)]
ordinaMosse dimensione scacchiera mosse =
  sortBy (comparing fst) $
  map (\pos ->
      (calcolaAccessibilita dimensione scacchiera pos, pos)
  ) mosse

{- Aggiorna la scacchiera con la nuova mossa.
- il primo argomento e' la scacchiera.
- il secondo argomento e' la posizione della mossa.
- il terzo argomento e' il numero della mossa corrente. -}
aggiornaScacchiera :: Scacchiera -> Posizione -> Int -> Scacchiera
aggiornaScacchiera scacchiera (x, y) mossa =
  take x scacchiera ++
  [take y (scacchiera !! x) ++
   [mossa] ++
   drop (y + 1) (scacchiera !! x)] ++
  drop (x + 1) scacchiera

{- Risolve il problema del giro del cavallo usando l'algoritmo di Warnsdorff-Squirrel.
- il primo argomento e' la dimensione della scacchiera.
- il secondo argomento e' la posizione di partenza del cavaliere. -}
risolviGiroCavallo :: Int -> Posizione -> IO (Maybe Scacchiera)
risolviGiroCavallo dimensione partenza = do
  riferimentoScacchiera <- newIORef (dimensione * dimensione, inizializzaScacchiera
dimensione)
  algoritmoWarnsdorffSquirrel dimensione (inizializzaScacchiera dimensione) partenza 1
  riferimentoScacchiera

{- Implementa l'algoritmo di Warnsdorff-Squirrel per risolvere il giro del cavallo.
- il primo argomento e' la dimensione della scacchiera.
- il secondo argomento e' la scacchiera corrente.
- il terzo argomento e' la posizione corrente del cavallo.
- il quarto argomento e' il numero della mossa corrente.
- il quinto argomento e' un riferimento IORef contenente lo stato delle caselle
mancanti e
la scacchiera. -}
algoritmoWarnsdorffSquirrel :: Int -> Scacchiera -> Posizione
-> Int -> IORef (Int, Scacchiera) -> IO (Maybe Scacchiera)
algoritmoWarnsdorffSquirrel dimensione scacchiera posizione mossa riferimentoScacchiera =
  do
    let scacchieraAggiornata = aggiornaScacchiera scacchiera posizione mossa
    let caselleMancanti = dimensione * dimensione - mossa
    writeIORef riferimentoScacchiera (caselleMancanti, scacchieraAggiornata)
    if mossa == dimensione * dimensione
      then return (Just scacchieraAggiornata)
    else do
      let prossimeMosse = filter (mossaValida dimensione scacchieraAggiornata) [(fst
posizione + dx, snd posizione + dy) | (dx, dy) <- mosseCavallo]
      if null prossimeMosse
        then return Nothing
```

```
        else do
            let mosseOrdinate = ordinaMosse dimensione scacchieraAggiornata
    prossimeMosse
            tentaMossa mosseOrdinate scacchieraAggiornata
where
    tentaMossa [] _ = return Nothing
    tentaMossa ((_, prossimaPosizione):resto) scacchieraCorrente = do
        risultato <- algoritmoWarnsdorffSquirrel dimensione
                                scacchieraCorrente
                                prossimaPosizione
                                (mossa + 1)
                                riferimentoScacchiera
        case risultato of
            Just soluzione -> return (Just soluzione)
            Nothing -> tentaMossa resto scacchieraCorrente

{- Stampa la scacchiera.
   - l'unico argomento e' la scacchiera da stampare. -}
stampaScacchiera :: Scacchiera -> IO ()
stampaScacchiera scacchiera = mapM_ (putStrLn . unwords . map (pad . show)) scacchiera
    where pad s = replicate (3 - length s) ' ' ++ s
```


File sorgente squirrelWarnsdorff.pl:

```
/* Programma Prolog per risolvere il problema del giro del cavallo usando l'algoritmo di
   Warnsdorff-Squirrel con ricorsione in coda. */

main :-
    leggi_dimensione_scacchiera(Dimensione),
    leggi_posizione(Dimensione, InizioX, InizioY),
    statistics(runtime, [TempoInizio|_]), % Inizia a monitorare il tempo di esecuzione
    write('Attendere la soluzione...'), nl,
    (   risolvi(Dimensione, (InizioX, InizioY), TempoInizio, ScacchieraFinale)
    -> stampa_scacchiera(ScacchieraFinale)
    ;   write('Soluzione non trovata.'), nl
    ).

/* Risolve il problema del giro del cavallo.
   - Dimensione: dimensione della scacchiera.
   - PosizioneIniziale: posizione iniziale del cavallo (InizioX, InizioY).
   - TempoInizio: tempo di inizio esecuzione.
   - ScacchieraFinale: la scacchiera finale con il percorso del cavallo.
*/
risolvi(Dimensione, PosizioneIniziale, TempoInizio, ScacchieraFinale) :-
    inizializza_scacchiera(Dimensione, Scacchiera),
    algoritmo_warnsdorff(Dimensione, Scacchiera, PosizioneIniziale, 1, TempoInizio,
        ScacchieraFinale).

/* Implementa l'algoritmo Warnsdorff-Squirrel con ricorsione in coda.
   - Dimensione: dimensione della scacchiera.
   - Scacchiera: scacchiera corrente.
   - (X, Y): posizione corrente del cavallo.
   - Mossa: numero della mossa corrente.
   - TempoInizio: tempo di inizio esecuzione.
   - ScacchieraFinale: la scacchiera finale con il percorso completato.
*/
algoritmo_warnsdorff(Dimensione, Scacchiera, (X, Y), Mossa, TempoInizio, ScacchieraFinale)
:-
    statistics(runtime, [TempoAttuale|_]), % Ottiene il tempo corrente
    TempoTrascorso is TempoAttuale - TempoInizio,
    (TempoTrascorso > 120000 ->
        format('Tempo limite superato: ~w ms\n', [TempoTrascorso]),
        halt % Interrompe l'esecuzione se supera 120 secondi (120000 ms)
    ; aggiorna_scacchiera(Scacchiera, (X, Y), Mossa, ScacchieraAggiornata),
        (   Mossa == Dimensione * Dimensione % Caso base: tutte le celle sono state
        visitate
        -> ScacchieraFinale = ScacchieraAggiornata
        ;   findall((NX, NY),
            (   mosse_cavallo((X, Y), (NX, NY)),
            mosse_valida(Dimensione, ScacchieraAggiornata, (NX, NY))
            ),
            Mosse),
            ordina_mosse(Dimensione, ScacchieraAggiornata, Mosse, MosseOrdinate),
            member(ProssimaMossa, MosseOrdinate),
            algoritmo_warnsdorff(Dimensione, ScacchieraAggiornata, ProssimaMossa,
                Mossa + 1, TempoInizio, ScacchieraFinale)
        )
    )
```

```
    ).

/* Legge e valida la dimensione della scacchiera.
*/
leggi_dimensione_scacchiera(Dimensione) :-
    write('Inserisci la dimensione della scacchiera (intero compreso tra 5 e 70): '),
    read(Ingresso),
    (   integer(Ingresso), Ingresso >= 5, Ingresso <= 70 ->
        Dimensione = Ingresso
    ;   (integer(Ingresso) ->
        write('Dimensione non valida. Deve essere compresa tra 5 e 70.')
        ;   write('Dato inserito non valido. Inserisci un numero intero.')
        ), nl, leggi_dimensione_scacchiera(Dimensione)
    ).

/* Legge e valida la posizione iniziale del cavallo.
*/
leggi_posizione(N, InizioX, InizioY) :-
    repeat,
    write('Inserisci la posizione di partenza del cavallo (X,Y)'),
    format(' (X e Y interi compresi tra 0 e ~d): ', [N-1]),
    catch(read_term(user_input, Ingresso, []), _, fallimento),
    (   analizza_ingresso(Ingresso, N, InizioX, InizioY)
    -> !
    ;   fallimento).

/* Funzione di fallimento chiamata in caso di errore durante la lettura
*/
fallimento :-
    write('Dato inserito non valido o errore di sintassi. Riprova.\n'),
    fail.

/* Analizza il dato in ingresso e verifica che sia nel formato corretto (X,Y).
*/
analizza_ingresso((X,Y), N, X, Y) :-
    integer(X), integer(Y), X >= 0, X < N, Y >= 0, Y < N.
analizza_ingresso(_, _, _, _) :-
    fail.

/* Inizializza la scacchiera con valori di default (-1).
*/
inizializza_scacchiera(N, Scacchiera) :-
    length(Scacchiera, N), % Crea una lista di N righe
    maplist(crea_riga(N), Scacchiera).

/* Crea una riga della scacchiera con valori di default (-1).
*/
crea_riga(N, Riga) :-
    length(Riga, N), % Crea una riga di lunghezza N
    maplist(= (-1), Riga). % Inizializza tutte le celle a -1

/* Genera le mosse possibili del cavallo a partire da una posizione.
*/
mosse_cavallo((X, Y), (NuovoX, NuovoY)) :-
    member((DeltaX, DeltaY), [(2, 1), (1, 2), (-1, 2), (-2, 1),
                              (-2, -1), (-1, -2), (1, -2), (2, -1)]),
```

```
NuovoX is X + DeltaX,
NuovoY is Y + DeltaY,
NuovoX >= 0,
NuovoY >= 0.

/* Controlla se una mossa e' valida (ossia se e' dentro i confini della scacchiera e la
   cella non e' stata visitata).
*/
mossa_valida(N, Scacchiera, (X, Y)) :-
    X >= 0, X < N, Y >= 0, Y < N,
    nth0(X, Scacchiera, Riga),
    nth0(Y, Riga, -1).

/* Calcola l'accessibilita' di una posizione (numero di mosse valide possibili).
*/
calcola_accessibilita_con_posizione(N, Scacchiera, (X, Y), Grado-(X, Y)) :-
    findall((NuovoX, NuovoY), (mosse_cavallo((X, Y), (NuovoX, NuovoY)),
        mossa_valida(N, Scacchiera, (NuovoX, NuovoY))), Mosse),
    length(Mosse, Grado).

/* Ordina le mosse in base all'accessibilita' (secondo la regola di Warnsdorff).
*/
ordina_mosse(N, Scacchiera, Mosse, MosseOrdinate) :-
    maplist(calcola_accessibilita_con_posizione(N, Scacchiera), Mosse,
        MosseConAccessibilita),
    keysort(MosseConAccessibilita, MosseOrdinateConAccessibilita),
    estrai_valori(MosseOrdinateConAccessibilita, MosseOrdinate).

/* Estrae i valori da una lista di coppie chiave-valore.
*/
estrai_valori([], []).
estrai_valori([_Val|Resto], [Val|ValoriResto]) :-
    estrai_valori(Resto, ValoriResto).

/* Aggiorna la scacchiera con una nuova mossa.
*/
aggiorna_scacchiera(Scacchiera, (X, Y), Mossa, NuovaScacchiera) :-
    (X >= 0, Y >= 0 ->
        nth0(X, Scacchiera, Riga),
        aggiorna_lista(Riga, Y, Mossa, NuovaRiga),
        aggiorna_lista(Scacchiera, X, NuovaRiga, NuovaScacchiera)
    ; throw(error(domain_error(not_less_than_zero, (X, Y)), _))).

/* Aggiorna una lista con un nuovo elemento in una posizione specifica.
*/
aggiorna_lista([_|Resto], 0, Elem, [Elem|Resto]) :- !.
aggiorna_lista([Testa|Resto], Indice, Elem, [Testa|NuovoResto]) :-
    (Indice > 0 ->
        NuovoIndice is Indice - 1,
        aggiorna_lista(Resto, NuovoIndice, Elem, NuovoResto)
    ; throw(error(domain_error(not_less_than_zero, Indice), _))).

/* Stampa la scacchiera.
*/
stampa_scacchiera(Scacchiera) :-
    stampa_righe(Scacchiera).
```

```
/* Stampa tutte le righe della scacchiera.
*/
stampa_righe([]).
stampa_righe([Riga|Resto]) :-
    stampa_riga(Riga),
    nl,
    stampa_righe(Resto).

/* Stampa una singola riga della scacchiera.
*/
stampa_riga([]).
stampa_riga([Elemento|Resto]) :-
    format('~d ', [Elemento]), % Stampa l'elemento con uno spazio
    stampa_riga(Resto).
```

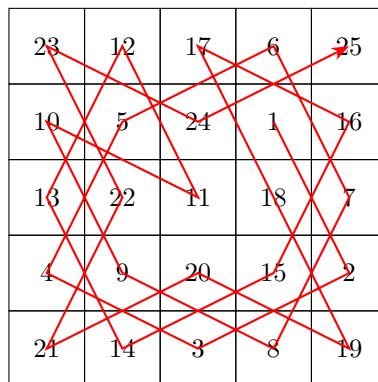
5 Testing del Programma

5.1 Test Haskell 1

Dimensione scacchiera: 5

Posizione iniziale cavallo: (1, 3)

Giro:

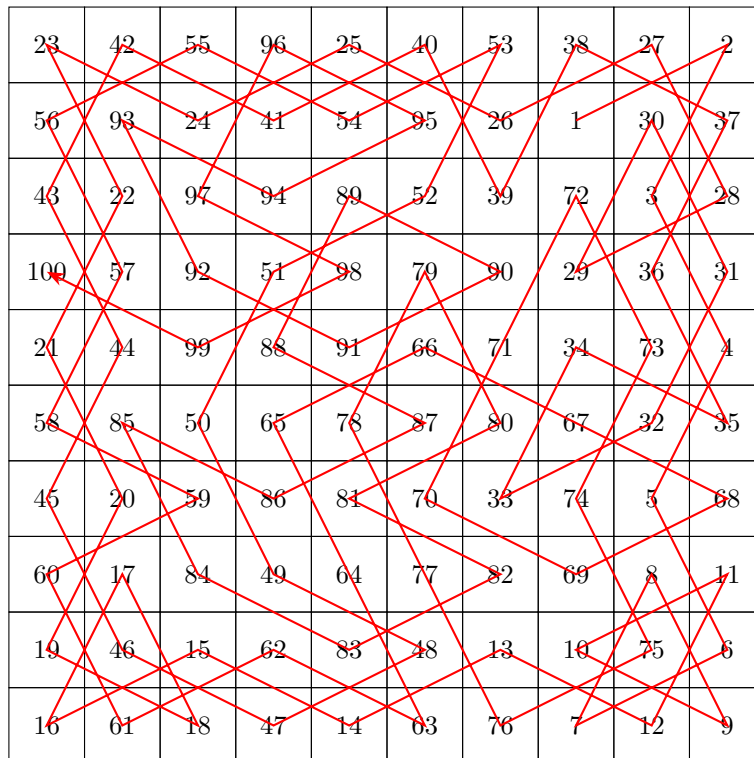


5.2 Test Haskell 2

Dimensione scacchiera: 10

Posizione iniziale cavallo: (1, 7)

Giro:



5.3 Test Haskell 3

Dimensione scacchiera: 8

Posizione iniziale cavallo: (0, 0)

Giro:

1	34	3	18	49	32	13	16
4	19	56	33	14	17	50	31
57	2	35	48	55	52	15	12
20	5	60	53	36	47	30	51
41	58	37	46	61	54	11	26
6	21	42	59	38	27	64	29
43	40	23	8	45	62	25	10
22	7	44	39	24	9	28	63

5.4 Test Haskell 4

Dimensione scacchiera: 7

Posizione iniziale cavallo: (3, 3)

Giro:

25	8	21	36	17	6	19
22	41	24	7	20	35	16
9	26	43	40	37	18	5
42	23	38	1	32	15	34
27	10	47	44	39	4	31
48	45	12	29	2	33	14
11	28	49	46	13	30	3

5.5 Test Haskell 5

Dimensione scacchiera: 7

Posizione iniziale cavallo: (6, 6)

Giro:

49	6	19	36	15	4	17
20	35	48	5	18	33	14
7	46	41	34	37	16	3
42	21	30	47	40	13	32
29	8	45	38	31	2	25
22	43	10	27	24	39	12
9	28	23	44	11	26	1

5.6 Test Haskell 6

Dimensione scacchiera: 7

Posizione iniziale cavallo: (0, 6)

Giro:

35	6	25	22	15	4	1
26	23	34	5	2	21	14
7	36	45	24	41	16	3
46	27	40	33	44	13	20
37	8	49	42	19	32	17
28	47	10	39	30	43	12
9	38	29	48	11	18	31

5.7 Test Haskell 7

Dimensione scacchiera: 7

Posizione iniziale cavallo: (6, 0)

Giro:

33	14	21	46	7	12	9
20	49	32	13	10	23	6
15	34	45	22	47	8	11
42	19	48	31	38	5	24
35	16	41	44	25	30	27
18	43	2	37	28	39	4
1	36	17	40	3	26	29

5.8 Test Haskell 8

Dimensione scacchiera: 9

Posizione iniziale cavallo: (2, 6)

Giro:

67	22	5	18	61	58	3	16	41
6	19	68	59	4	17	42	57	2
23	66	21	62	79	60	1	40	15
20	7	78	69	54	39	56	43	52
71	24	65	80	63	76	53	14	45
8	81	70	77	38	55	44	51	34
25	72	27	64	75	50	33	46	13
28	9	74	37	30	11	48	35	32
73	26	29	10	49	36	31	12	47

5.9 Test Haskell 9

Dimensione scacchiera: 11

Posizione iniziale cavallo: (3, 7)

Giro:

71	82	23	110	69	80	25	104	67	78	27
22	121	70	81	24	109	68	79	26	99	66
83	72	113	118	111	74	105	100	103	28	77
114	21	120	73	106	101	108	1	76	65	98
43	84	117	112	119	94	75	102	97	2	29
20	115	44	85	92	107	96	59	32	57	64
45	42	89	116	95	50	93	56	63	30	3
88	19	46	41	86	91	60	31	58	33	54
13	16	87	90	49	38	51	62	55	4	7
18	47	14	11	40	61	36	9	6	53	34
15	12	17	48	37	10	39	52	35	8	5

5.10 Test Haskell 10

Dimensione scacchiera: 14

Posizione iniziale cavallo: (2, 8)

Giro:

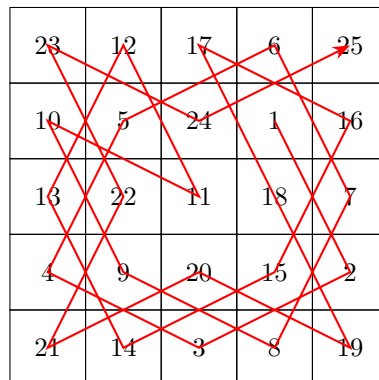
31	146	169	56	33	144	189	54	35	2	139	52	37	4
168	57	32	145	196	55	34	143	192	53	36	3	40	51
147	30	167	170	149	188	193	190	1	138	77	140	5	38
58	171	148	181	166	195	150	185	142	191	48	39	50	41
29	154	165	172	177	184	187	194	137	78	141	76	47	6
156	59	180	153	182	151	176	107	186	105	112	49	42	75
131	28	155	164	173	178	183	136	111	108	79	104	7	46
60	157	132	179	152	161	110	175	106	113	92	45	74	43
27	130	127	160	163	174	135	114	109	90	103	80	71	8
128	61	158	133	126	115	162	101	124	93	88	91	44	73
119	26	129	98	159	134	125	94	89	102	81	72	9	70
62	23	118	121	116	97	100	123	82	87	68	85	12	15
25	120	21	64	99	122	19	66	95	84	17	14	69	10
22	63	24	117	20	65	96	83	18	67	86	11	16	13

5.11 Test Prolog 1

Dimensione scacchiera: 5

Posizione iniziale cavallo: (1, 3)

Giro:

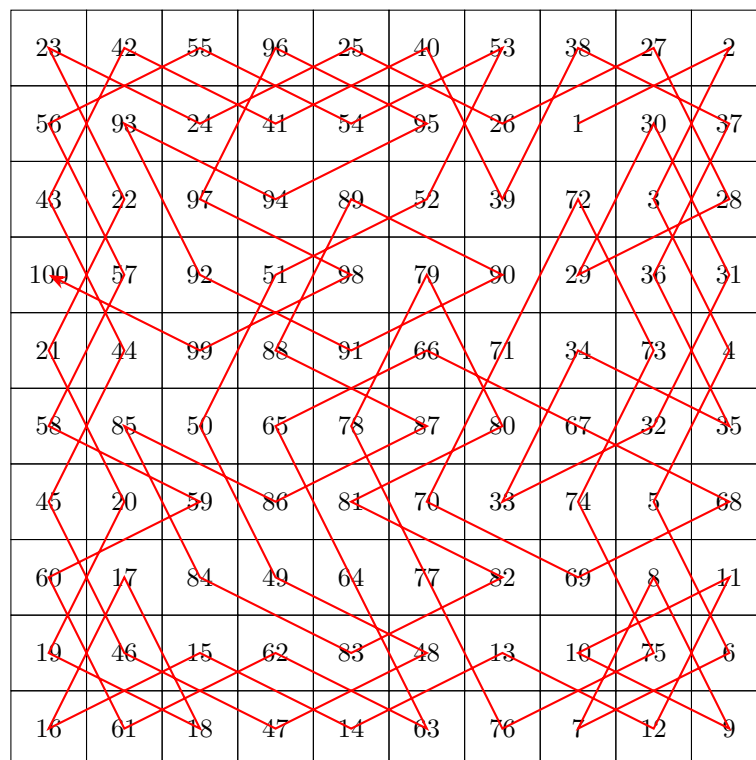


5.12 Test Prolog 2

Dimensione scacchiera: 10

Posizione iniziale cavallo: (1, 7)

Giro:



5.13 Test Prolog 3

Dimensione scacchiera: 8

Posizione iniziale cavallo: (0, 0)

Giro:

1	34	3	18	49	32	13	16
4	19	56	33	14	17	50	31
57	2	35	48	55	52	15	12
20	5	60	53	36	47	30	51
41	58	37	46	61	54	11	26
6	21	42	59	38	27	64	29
43	40	23	8	45	62	25	10
22	7	44	39	24	9	28	63

5.14 Test Prolog 4

Dimensione scacchiera: 7

Posizione iniziale cavallo: (3, 3)

Giro:

25	8	21	36	17	6	19
22	41	24	7	20	35	16
9	26	43	40	37	18	5
42	23	38	1	32	15	34
27	10	47	44	39	4	31
48	45	12	29	2	33	14
11	28	49	46	13	30	3

5.15 Test Prolog 5

Dimensione scacchiera: 7

Posizione iniziale cavallo: (6, 6)

Giro:

49	6	19	36	15	4	17
20	35	48	5	18	33	14
7	46	41	34	37	16	3
42	21	30	47	40	13	32
29	8	45	38	31	2	25
22	43	10	27	24	39	12
9	28	23	44	11	26	1

5.16 Test Prolog 6

Dimensione scacchiera: 7

Posizione iniziale cavallo: (0, 6)

Giro:

35	6	25	22	15	4	1
26	23	34	5	2	21	14
7	36	45	24	41	16	3
46	27	40	33	44	13	20
37	8	49	42	19	32	17
28	47	10	39	30	43	12
9	38	29	48	11	18	31

5.17 Test Prolog 7

Dimensione scacchiera: 7

Posizione iniziale cavallo: (6, 0)

Giro:

33	14	21	46	7	12	9
20	49	32	13	10	23	6
15	34	45	22	47	8	11
42	19	48	31	38	5	24
35	16	41	44	25	30	27
18	43	2	37	28	39	4
1	36	17	40	3	26	29

5.18 Test Prolog 8

Dimensione scacchiera: 9

Posizione iniziale cavallo: (2, 6)

Giro:

67	22	5	18	61	58	3	16	41
6	19	68	59	4	17	42	57	2
23	66	21	62	79	60	1	40	15
20	7	78	69	54	39	56	43	52
71	24	65	80	63	76	53	14	45
8	81	70	77	38	55	44	51	34
25	72	27	64	75	50	33	46	13
28	9	74	37	30	11	48	35	32
73	26	29	10	49	36	31	12	47

5.19 Test Prolog 9

Dimensione scacchiera: 11

Posizione iniziale cavallo: (3, 7)

Giro:

71	82	23	110	69	80	25	104	67	78	27
22	121	70	81	24	109	68	79	26	99	66
83	72	113	118	111	74	105	100	103	28	77
114	21	120	73	106	101	108	1	76	65	98
43	84	117	112	119	94	75	102	97	2	29
20	115	44	85	92	107	96	59	32	57	64
45	42	89	116	95	50	93	56	63	30	3
88	19	46	41	86	91	60	31	58	33	54
13	16	87	90	49	38	51	62	55	4	7
18	47	14	11	40	61	36	9	6	53	34
15	12	17	48	37	10	39	52	35	8	5

5.20 Test Prolog 10

Dimensione scacchiera: 14

Posizione iniziale cavallo: (2, 8)

Giro:

31	146	169	56	33	144	189	54	35	2	139	52	37	4
168	57	32	145	196	55	34	143	192	53	36	3	40	51
147	30	167	170	149	188	193	190	1	138	77	140	5	38
58	171	148	181	166	195	150	185	142	191	48	39	50	41
29	154	165	172	177	184	187	194	137	78	141	76	47	6
156	59	180	153	182	151	176	107	186	105	112	49	42	75
131	28	155	164	173	178	183	136	111	108	79	104	7	46
60	157	132	179	152	161	110	175	106	113	92	45	74	43
27	130	127	160	163	174	135	114	109	90	103	80	71	8
128	61	158	133	126	115	162	101	124	93	88	91	44	73
119	26	129	98	159	134	125	94	89	102	81	72	9	70
62	23	118	121	116	97	100	123	82	87	68	85	12	15
25	120	21	64	99	122	19	66	95	84	17	14	69	10
22	63	24	117	20	65	96	83	18	67	86	11	16	13