Laboratorio di Algoritmi e Strutture Dati

Docenti: M. Torelli, S. Aguzzoli

Appelli di giugno e luglio 2009
Progetto “Hitori”

Introduzione

Il progetto è ispirato ad un rompicapo logico di origine giapponese, chiamato *hitori*. L'hitori è costituito da una griglia quadrata di caselle, ciascuna delle quali contiene un numero. Scopo del gioco è quello di annerire alcune caselle, in modo da soddisfare alcune proprietà che verranno meglio definite in seguito.

Obiettivo del progetto è sviluppare un programma per verificare la correttezza di soluzioni di griglie hitori e/o applicare tecniche risolutive. Si richiede allo studente di effettuare un **adeguato collaudo** del proprio progetto su numerosi esempi diversi per verificarne la correttezza.

La realizzazione del progetto è una prova d'esame da svolgersi **individualmente**. I progetti giudicati frutto di **copiatura** saranno **estromessi** d'ufficio dalla valutazione.

La versione aggiornata del progetto è pubblicata in `.pdf` sul sito: `http://aguzzoli.dsi.unimi.it/algo/`. Si consiglia di consultare periodicamente questo sito per eventuali correzioni e/o precisazioni relative al testo del progetto.

Organizzazione degli appelli e modalità di consegna

Il presente progetto è valido per gli appelli di giugno e luglio 2009 e deve essere consegnato:

- entro il 22 giugno compreso, per l'appello di giugno;
- oppure entro il 13 luglio compreso, per l'appello di luglio.

Le discussioni dei progetti per i due appelli si svolgeranno in date e luoghi da specificarsi, comunque entro le date di svolgimento delle prove orali. Il calendario degli esami orali sarà disponibile sulla pagina del corso `http://aguzzoli.dsi.unimi.it/algo` qualche giorno dopo il termine di consegna del progetto.

Il progetto va inviato per posta elettronica all'indirizzo `aguzzoli@dsi.unimi.it` entro le date sopra indicate. Occorre presentare:

1. il codice sorgente (rigorosamente ANSI C, compilabile con **gcc**);
2. una sintetica relazione (formato pdf o rtf) che illustra le strutture dati utilizzate e le scelte implementative, analizzando il costo delle diverse operazioni richieste dalla specifica.

I due o più file (file sorgenti C + relazione) devono essere contenuti in un unico file `.zip` il cui nome dovrà essere della forma `cognome_matricola.zip`. La relazione e il codice devono riportare nome, cognome e matricola. Una copia cartacea della relazione e del codice deve inoltre essere consegnata al Dott. Aguzzoli entro le scadenze fissate (lasciandola eventualmente nella sua casella postale presso il dipartimento in via Comelico). Si ricorda infine di presentarsi alla prova orale con una copia stampata della relazione e del codice.

Regole del gioco

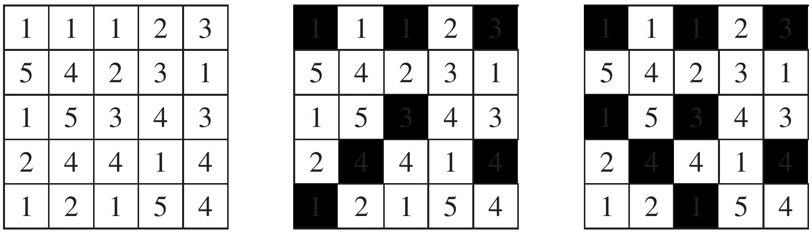
Un Hitori è costituito da una griglia quadrata di caselle, ciascuna delle quali contiene un numero. La dimensione della griglia può variare; detto n il suo lato, i numeri contenuti nella griglia variano tra 1 e n (inclusi).

Scopo del gioco è quello di annerire alcune caselle (cancellando i numeri che esse contengono), in modo da soddisfare le seguenti tre proprietà:

1. ogni numero deve comparire al più una volta per ogni riga e per ogni colonna;
2. due caselle nere possono avere al massimo un vertice comune (cioè possono essere adiacenti in diagonale ma non in orizzontale nè in verticale);
3. le caselle non annerite devono formare una sola componente connessa verticalmente e orizzontalmente, ovvero non devono esserci caselle o gruppi di caselle isolate dal resto dello schema.

Chiamiamo *soluzione* una qualsiasi configurazione di caselle nere; una soluzione è *corretta* se soddisfa le precedenti tre proprietà. Nel seguito considereremo soltanto hitori che ammettono **una e una sola soluzione corretta**.

Nella seguente figura è illustrato un esempio di Hitori di dimensione 5×5: a sinistra la griglia da risolvere, al centro la sua soluzione corretta, a destra un esempio di soluzione non corretta (le caselle nell'angolo in basso a sinistra sono isolate dal resto della griglia, quindi la proprietà 3 non è verificata).



Tecniche di risoluzione

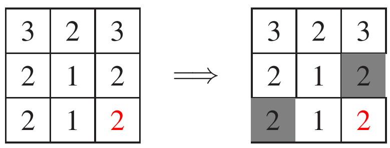
In linea teorica, la soluzione corretta di un hitori può essere trovata considerando tutte le possibili soluzioni ed escludendo tutte quelle che non sono corrette. Chiaramente questo metodo non può essere usato per risolvere un hitori a mano, e anche usando un calcolatore può diventare molto oneroso con l'aumentare delle dimensioni della griglia.

Tuttavia, le possibilità possono essere drasticamente ridotte usando alcune euristiche, basate sulla ricerca di *pattern* che implicano necessariamente che una certa cella **debba** essere annerita o, viceversa, restare bianca.

Citiamo qui alcune di queste tecniche, illustrandole con degli esempi: per facilitare la lettura delle griglie, anziché annerire le caselle usiamo un fondo grigio e indichiamo in rosso i numeri che non possono essere anneriti; inoltre, quando i numeri non sono rilevanti, usiamo $*$ per indicare un numero non meglio determinato.

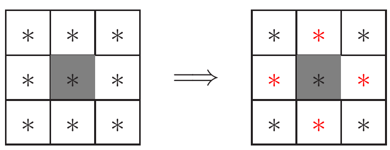
[SB] Solo un bianco

Se si stabilisce che una casella non può essere annerita, e nella stessa riga o nella stessa colonna si trova un altro numero uguale, allora questo deve essere annerito (altrimenti la regola 1 non sarebbe rispettata).



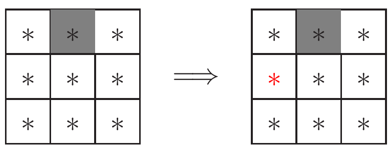
[NV] Nero vicino

Una volta annerita una casella, ne deriva che tutte le altre ad essa adiacenti non potranno essere annerite (per la regola 2).



[A] Angolo

Se è stata annerita una casella vicino ad un angolo della griglia, allora una casella adiacente in diagonale non può essere annerita (altrimenti la casella nell'angolo resterebbe isolata, contro la regola 3).



[PI] Panino imbottito

Se un numero si trova tra due numeri uguali, allora non può essere annerito (se venisse annerito, entrambi i suoi vicini non potrebbero essere anneriti per la regola 2, quindi avremmo un numero ripetuto nella stessa riga/colonna, contraddicendo la regola 3).

2	*	2	$\Longrightarrow$	2	*	2

[CS] Coppia e single

Se due numeri uguali sono adiacenti e la riga (o colonna) su cui si trovano contiene un altro numero uguale a loro, quest'ultimo dovrà essere annerito (per la regola 2, uno solo degli altri due può essere annerito).

3		3
*		*
3	$\Longrightarrow$	3
3		3

[T] Trapezio

Se si trova una configurazione a "trapezio", ci sono due caselle che non possono essere annerite, come in figura (se una di queste fosse annerita, allora i due numeri in diagonale non potrebbero essere anneriti per la regola 2, quindi entrambi i numeri adiacenti andrebbero anneriti per la regola 1, ma questo sarebbe impossibile per la regola 2).

2	3		2	3
*	*		*	*
2	*	$\Longrightarrow$	2	*
*	3		*	3

[BI] Bianco impossibile

Se forzando una casella a rimaner bianca e applicando ricorsivamente le tecniche SB e NV si ottiene necessariamente una configurazione che non rispetta le tre regole, allora la casella deve essere annerita necessariamente.

[NI] Nero impossibile

Se annerendo una casella e applicando ricorsivamente le tecniche NV e SB si ottiene necessariamente una configurazione che non rispetta le tre regole, allora la casella non può essere annerita.

Si noti che le ultime due tecniche si basano sostanzialmente su *dimostrazioni per assurdo*: assumo che la casella sia nera/bianca e ne ricavo una contraddizione. Naturalmente queste tecniche possono essere generalizzate in molti modi, ad esempio applicando ricorsivamente altre tecniche oltre a SB e NV.

Esempio

Illustriamo come possono essere applicate alcune delle tecniche precedenti per costruire la soluzione della griglia hitori proposta a pagina 2.

Ad ogni passaggio individuiamo una casella per la quale sappiamo dire con certezza se **deve** oppure **non può** essere annerita, scrivendo "si" o "no" rispettivamente, e indichiamo la sigla corrispondente alla tecnica che stiamo applicando per trarre questa conclusione.

Per individuare le caselle della griglia, usiamo l'usuale sistema di riferimento cartesiano: ogni casella è rappresentata da una coppia (i, j), dove $0 \leq i, j, \leq n-1$, essendo n la dimensione della griglia; la coppia (i, j) indica la casella nella riga i (a partire dall'alto) e nella colonna j (a partire da sinistra). Ad esempio, la casella in alto a sinistra sarà indicata da $(0,0)$.

no (0, 1) : PI

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (1, 0) : NV

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (1, 2) : NV

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (2, 3) : PI

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (4, 1) : PI

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (2, 1) : PI

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

si (3, 4) : CS

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

si (0, 0) : SB

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

si (0, 2) : SB

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (0, 3) : NV

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (3, 3) : PI

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (1, 4) : PI

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (3, 0) : PI

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (2, 4) : NV

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

si (2, 2) : SB

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (3, 2) : NV

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

si (3, 1) : SB

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

si (0, 4) : SB

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (4, 4) : NV

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (4, 3) : A

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

no (4, 2) : NI

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

si (4, 0) : SB

1	1	1	2	3
5	4	2	3	1
1	5	3	4	3
2	4	4	1	4
1	2	1	5	4

Specifiche di implementazione

Il programma dovrà innanzitutto leggere da standard input (`stdin`) una griglia hitori (con alcune caselle eventualmente già annerite). La griglia hitori in input è rappresentata da un intero n seguito da una sequenza di n righe (separate da `\n`), ciascuna delle quali è a sua volta composta da una sequenza di n interi separati da spazi. Eventuali caselle già annerite saranno rappresentate dal numero 0.

Se la griglia contiene almeno uno 0, allora il programma dovrà semplicemente verificare che la soluzione proposta sia corretta. In caso contrario dovrà **provare** a risolvere l'hitori fornito in input, usando ad esempio le tecniche proposte sopra.

La costruzione della soluzione deve essere descritta come una sequenza di passaggi, in ciascuno del quali si individua una casella per la quale si può dire con certezza se **deve** oppure **non può** essere annerita. Per ogni passaggio deve essere stampata una riga della forma

```
XX (i, j) : YY
```

dove (i, j) indica la casella della quale si stabilisce se deve o non può essere annerita, `XX` è la stringa `si` oppure `no` a seconda che la casella debba oppure non possa essere annerita, e `YY` è una sigla corrispondente alla tecnica applicata. L'output deve quindi concludersi con la soluzione trovata (rappresentata sostituendo 0 al posto dei numeri da annerire).

E' naturalmente possibile implementare anche altre tecniche risolutive, basate ad esempio sul backtracking. In questo caso, l'applicazione delle tecniche deve essere documentata in output in modo analogo a quanto definito sopra e si devono fornire nella relazione le indicazioni necessarie ad interpretare l'output.

In generale, la valutazione del progetto dipenderà dalla capacità del programma di risolvere hitori efficientemente. **Saranno tuttavia ammessi alla valutazione anche progetti che non sono in grado di trovare una soluzione per tutte le griglie, purchè sappiano verificare la correttezza di griglie già risolte, implementino almeno le tecniche SB, NV, PI, CS, A per quanto riguarda la consegna di giugno, e anche le rimanenti tecniche per quanto riguarda la consegna di luglio.** In questo caso, la valutazione si baserà sul numero di tecniche implementate, sulla qualità della loro implementazione, nonchè sugli algoritmi usati per combinare le loro applicazioni.

Esempi di esecuzione

Consideriamo la griglia hitori e le mosse presentate nell'esempio di pagina 3. Ad esse corrisponderebbero i seguenti input e output.

Input

```
5
1 1 1 2 3
5 4 2 3 1
1 5 3 4 3
2 4 4 1 4
1 2 1 5 4
```

Output

```
no (0, 1) : PI
si (0, 0) : SB
no (1, 0) : NV
si (0, 2) : SB
no (1, 2) : NV
no (0, 3) : NV
no (2, 3) : PI
no (3, 3) : PI
no (4, 1) : PI
no (1, 4) : PI
no (2, 1) : PI
no (3, 0) : PI
si (3, 4) : CS
no (2, 4) : NV
si (2, 2) : SB
no (3, 2) : NV
si (3, 1) : SB
si (0, 4) : SB
no (4, 4) : NV
no (4, 3) : A
no (4, 2) : NI
si (4, 0) : SB

0 1 0 2 0
5 4 2 3 1
1 5 0 4 3
2 0 4 1 0
0 2 1 5 4
```

Consideriamo sempre la stessa griglia hitori ma forniamola in input risolta (usando 0 per indicare le caselle nere).

Input

```
5
1 0 3 4 2
2 4 5 0 1
3 0 4 5 0
5 3 1 2 4
0 1 0 3 0
```

Output

```
SOLUZIONE CORRETTA
```

Suggerimenti

Applicazione ricorsiva di NV, A e SB

Osservate che ogni volta che si decide che una casella va annerita, tutte le caselle vicine dovranno restare bianche per la tecnica NV e, se la casella annerita si trova in angolo, ulteriori caselle dovranno restare bianche per la tecnica A. Viceversa, quando si decide che un numero non può essere annerito, allora tutti i numeri uguali nella stessa riga e nella stessa colonna andranno anneriti per la tecnica SB. Questo suggerisce di scrivere due funzioni del tipo

```
annerisci( ... i, ... j,  ... )
non_annerire( ... i, ... j,  ... )
```

che, oltre a colorare la cella (i, j) del colore opportuno, applicano poi ricorsivamente le regole NV e A oppure SB richiamando rispettivamente le funzioni `non_annerire` o `annerisci` su altre caselle della griglia.

Dove reperire altri esempi

Sul Web esistono molti siti che propongono griglie hitori risolte o da risolvere, a cui potete fare riferimento per acquisire dimistichezza con le regole del gioco, per scoprire nuove tecniche risolutive e, soprattutto, per reperire griglie hitori da usare per testare il vostro programma. Una buona fonte è ad esempio il sito

`http://www.menneske.no/hitori/eng/`

Colorazione dell'output

Anche se non è richiesto dalla specifiche di output, in fase di sviluppo e di *debugging* è opportuno prevedere la stampa, passaggio dopo passaggio, della griglia hitori aggiornata, come nell'esempio di pagina 3. Per rappresentare le caselle annerite e quelle che non possono esserlo, è possibile usare varie notazioni (ad esempio `*3*` può indicare che il 3 è stato annerito, `-3-` che il 3 non andrà mai annerito, `3` che ancora non si è stabilito se questa casella va annerita oppure no).

Per migliorare la leggibilità delle griglie, è possibile usare le *ANSI escape sequences* che, tra le altre cose, permettono di modificare alcuni aspetti grafici del terminale, come i colori dei caratteri o dello sfondo. Si tratta di sequenze di caratteri ASCII della forma

```
Esc[...m
```

dove `Esc` è il carattere di Escape, con codice ASCII 27, e al posto dei puntini va sostituito un codice alfanumerico che controlla l'aspetto del terminale. Ad esempio, la sequenze `Esc[31m` permette di scrivere in rosso e la sequenza `Esc[40m` permette di scrivere su sfondo nero. All'indirizzo

`http://ascii-table.com/ansi-escape-sequences.php`

potete trovare l'elenco competo dei codici alfanumerici che si possono utilizzare.

Le *ANSI escape sequences* possono essere usate anche all'interno di programmi C, rappresentando opportunamente il carattere `Esc`. Con gcc, questo si può fare usando `\e`, ad esempio questa porzione di codice:

```
printf( "\e[31m ROSSO \e[0m" );
printf( "\e[40;37m SFONDO NERO \e[0m" );
```

stampa la stringa `ROSSO` in grassetto rosso su sfondo bianco seguita dalla stringa `SFONDO NERO` in caratteri bianchi su sfondo nero. Nota che la sequenze `\e[0m` permette di ripristinare le impostazioni standard del terminale.

Nota bene: con altri compilatori e in ambiente non UNIX non è detto che questa porzione di codice funzioni!