Macchine di Turing in due dimensioni

Lorenzo Repetto
Istituto Tecnico Industriale Statale "Italo Calvino" – Genova
Via Borzoli 21, 16153, repetto@calvino.ge.it

Parole chiave: Macchine di Turing, *array* a due indici, immagini bidimensionali, paradigma di programmazione *object oriented*.
Classi destinatarie dell'iniziativa didattica: quarte dell'indirizzo informatico.

L'esperienza qui presentata trae ispirazione da un vecchio lavoro di Greg Turk, dell'Università di Chapel Hill, North Carolina; ma l'idea di progettare e realizzare poi in laboratorio, secondo le metodologie più moderne, un programma che lo emulasse è dello scorso autunno, quando la proposi ai miei quindici studenti del quinto anno dell'indirizzo sperimentale "Abacus" come esercitazione di ripasso, con l'obiettivo di rivisitare alcune delle tematiche affrontate l'anno precedente. Il programma è stato sviluppato in linguaggio C++ (Microsoft Visual Studio 6.0).

Anzitutto, cerchiamo di descrivere in modo rigoroso le *classi di oggetti* che intendiamo trattare. Una *Macchina di Turing in due dimensioni* (MdT2D) è un automa in grado di muoversi su una tavola piana, suddivisa in celle quadrate di vari colori: pensiamo a una grande scacchiera con le case colorate, una delle quali è occupata da un automa che si può muovere di una sola casa per volta ortogonalmente, magari dopo aver cambiato il colore della casa lasciata…

La tavola dovrebbe estendersi senza limiti, ricoprendo interamente un piano infinito, ma noi supponiamo che abbia al più `MAX_RIGHE` righe e `MAX_COLONNE` colonne, essendo `MAX_RIGHE` e `MAX_COLONNE` due costanti prefissate. Ogni cella ha un *colore*, rappresentato da un intero >= `0` e < `num_colori`, essendo `num_colori` (<= `MAX_COLORI`, costante prefissata) il numero di colori (che giocano il ruolo dei simboli dell'alfabeto di una classica MdT) riconosciuti dalla macchina. Ad ogni passo della computazione, prima di fermarsi, la macchina si trova in uno *stato*, rappresentato da un intero >= `0` e < `num_stati`, essendo `num_stati` (<= `MAX_STATI`, costante prefissata) il numero di stati "attivi" possibili per la macchina; ogni eventuale stato >= `num_stati` sarà considerato "finale".

La macchina si trova inizialmente, per convenzione, nello *stato zero* (`0`), in una data cella della tavola, e da lì inizia la sua computazione, seguendo le regole espresse dalla sua *tabella di triple* (di `num_stati` righe per `num_colori` colonne): se la macchina è nello stato *s* e il colore della cella in cui si trova è *c* e la tripla di riga *s* e colonna *c* nella sua tabella è (*c1*, *sposta*, *s1*), allora la macchina cambia in *c1* il colore della cella in cui si trova, poi si sposta come stabilito da *sposta* (= `0` sta ferma, = `1` si muove di una cella verso l'alto, = `2` si muove di una cella a sinistra, = `3` si muove di una cella verso il basso, = `4` si muove di una cella a destra) e infine si porta nel suo nuovo stato *s1*. La computazione termina quando si verifica almeno una delle seguenti condizioni:

1. la macchina arriva in uno stato >= `num_stati`;

2. la macchina “esce” dalla tavola;
3. la macchina ha eseguito `MAX_PASSI` (costante prefissata) passi.

Quindi, a differenza delle classiche MdT (che si spostano lungo un nastro monodimensionale, ma potenzialmente infinito, e senza alcun limite al numero di passi), qui la computazione termina sempre. Quando ciò avviene, la tavola (che costituisce il risultato della computazione) può essere visualizzata come immagine bidimensionale, ad esempio facendo corrispondere una cella a un *pixel* e mappando il numero che rappresenta il colore della cella in una opportuna tavolozza di colori (o, più semplicemente, scala di grigi).

Abbiamo realizzato un programma che permette di simulare il comportamento di una MdT2D su una tavola. Dapprima abbiamo definito una classe `MdT2D` con:

- tre *campi privati*: `tabella` (matrice `[MAX_STATI][MAX_COLORI]` di *triple*, come illustrato sopra), `num_stati` e `num_colori` (che dicono, rispettivamente, quante righe e colonne di tale matrice sono effettivamente usate);
- un *costruttore*, che permette di inizializzare in modo appropriato tali campi, leggendo i dati da un *file* di testo specificato (si veda l’esempio sotto riportato);
- due *metodi getter*: uno che ritorna il valore del campo `num_stati` e un altro che, dati lo stato corrente *s* e il colore della cella corrente *c*, ritorna la tripla di riga *s* e colonna *c* in `tabella`.

L’importante è che in `tabella` siano assegnate *tutte* le triple utili, in numero `num_stati` × `num_colori`; sicché abbiamo pensato di avere i dati necessari in un *file* di testo – per fare un esempio, con le nove linee nel riquadro a sinistra:

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

stato \ colore	0	1
0	(1, 1, 1)	(0, 3, 3)
1	(1, 2, 2)	(0, 4, 0)
2	(1, 3, 3)	(0, 1, 1)
3	(1, 4, 0)	(0, 2, 2)

e di servircene per costruire l’istanza della classe `MdT2D` che rappresenta la MdT2D con quattro stati (`0`, `1`, `2`, `3`), due colori (`0`, `1`) e la tabella 4×2 le cui triple sono elencate, per righe, nel *file* (si veda il riquadro a destra nella figura sopra). Come dovrebbe essere ormai chiaro, se questa macchina si venisse a trovare nello stato `3`, sopra una cella di colore `0`, cambierebbe il colore di tale cella in `1`, si sposterebbe di una cella a destra (poiché la seconda componente della tripla è `4`) e passerebbe nello stato `0`… Qui le triple non prevedono stati “finali” (>= `4`): infatti ciascuna tripla rimanda in uno degli stati “attivi” (da `0` a `3`).

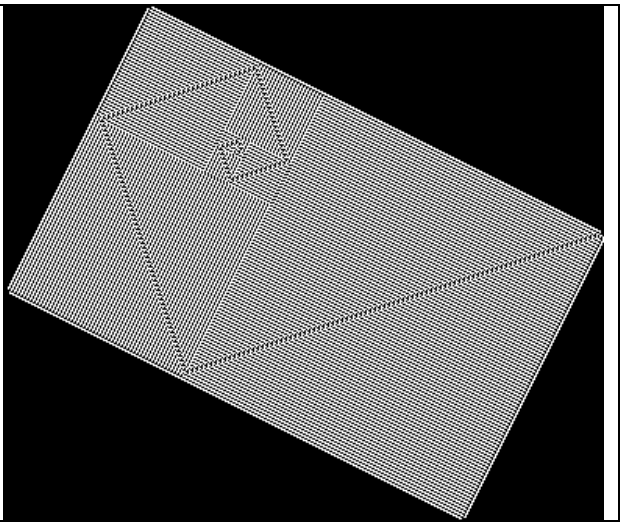
Abbiamo poi definito una classe `Tavola` con:

- tre *campi privati*: `tavola` (matrice `[MAX_RIGHE][MAX_COLONNE]` di *celle colorate*, come illustrato sopra), `num_righe` e `num_colonne` (che dicono, rispettivamente, quante righe e colonne di tale matrice sono effettivamente considerate);

- un *costruttore*, che permette di inizializzare in modo appropriato tali campi (per quanto riguarda le celle, assegna il colore `0` a tutte quelle usabili);
- un *metodo* con prototipo `void applica (MdT2D m, int i, int j)`, che innesca sulla tavola la computazione della macchina `m` (nello stato `0`) a partire dalla cella di riga `i` e colonna `j` (purché sia dentro la tavola!) e termina quando si verifica almeno una delle condizioni già illustrate sopra (abbiamo assunto, come *precondizione*, che inizialmente ciascuna cella della tavola abbia uno dei colori riconosciuti dalla macchina `m`, tra i quali certamente c'è lo `0`);
- un *metodo* che consente di salvare la tavola come immagine bidimensionale (su un *file* in formato PPM), mappando i colori delle celle su una scala di grigi.

Come esempio di risultato ottenuto, riportiamo qui sotto il contenuto di un *file* che specifica una MdT2D e l'immagine (in bianco e nero) risultante dalla sua applicazione a una tavola 330×385 inizialmente tutta colorata in nero (colore `0`), a partire dalla cella di riga 93 e colonna 150 (le righe sono numerate progressivamente dall'alto, le colonne da sinistra, sempre partendo da zero).

```
8 2
1 2 1
0 1 4
1 3 2
0 2 5
1 4 3
0 3 6
1 1 0
0 4 7
1 4 3
1 4 3
1 1 0
1 1 0
1 2 1
1 2 1
1 3 2
1 3 2
```



Questa macchina genera una configurazione "a spirale", evidenziando in particolare le diagonali di quadrati disposti via via in modo ordinato intorno al punto di partenza: pare che nell'immagine (ottenuta in 160741 passi, dopodiché la macchina esce dalla tavola) s'affacci la successione di Fibonacci!

Lasciamo scoprire a voi lettori l'altrettanto curioso disegno generato dalla macchina del primo esempio che abbiamo riportato, insieme col compito di progettare un'operazione per osservare sullo schermo l'immagine man mano che si forma, mentre la macchina esegue (a velocità opportuna!) i suoi passi di calcolo.