Homework 1: Sudoku

L'homework richiede di progettare ed implementare in Java un algoritmo parallelo per il conteggio delle soluzioni di istanze di Sudoku e di realizzare un'analisi sperimentale della soluzione proposta.

Il Sudoku. Si tratta di un puzzle di logica sorprendentemente semplice, introdotto in Giappone nel 1984 con il nome

SUuji wa DOKUshin ni kagiru

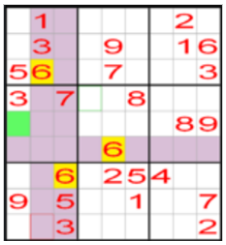
Nome troppo complesso da ricordare! E quindi ben presto abbreviato in *Sudoku*, ovvero "numero singolo": *su* vuol dire infatti "numero" e *doku* significa "singolo".

Il gioco avviene su una scacchiera 9×9, a sua volta divisa in 9 blocchi di dimensioni 3×3, e prevede di riempire la scacchiera posizionando le cifre da 1 a 9 in modo che nessuna cifra sia mai ripetuta in una stessa riga, una stessa colonna e uno stesso blocco. Ad esempio, data l'istanza nella scacchiera di sinistra, la scacchiera di destra mostra una possibile soluzione:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Semplici tecniche risolutive. Una tecnica molto naturale per risolvere il Sodoku prevede di calcolare, per ogni numero e per ogni blocco, tutte e sole le posizioni possibili di quel numero nel blocco. Ad esempio, considerate nella seguente griglia il blocco contenente la cella verde (prima colonna, sotto al numero 3):



In tale blocco, il numero 6 può trovarsi solo nella prima colonna: la seconda e la terza sono infatti impossibilitate a contenerlo per via della posizione del 6 rispettivamente nel blocco superiore e in quello inferiore. Inoltre il numero 6 può trovarsi solo nella prima o nella seconda riga: la terza è infatti impossibilitata a contenerlo per via della posizione del 6 nel blocco centrale. L'intersezione delle posizioni libere nelle prime due righe e nella prima colonna ci restituisce la cella verde, che quindi dovrà obbligatoriamente contenere il numero 6.

Una strategia simmetrica prevede di calcolare, per ogni cella vuota, l'insieme dei possibili ***candidati*** ad occupare quella cella. Ad esempio, i candidati per la cella verde sono tutti i numeri tranne 3 (già contenuto nel blocco), 5 (già presente sulla colonna), 7 (già contenuto nel blocco), 8 (già presente sulla riga) e 9 (già presente su riga e colonna). Rimangono quindi i numeri 1, 2, 4 e 6. Laddove una cella abbia un unico candidato, quella sarà obbligatoriamente la sua posizione.

Spazio delle soluzioni. Data un'istanza di Sudoku, il ***prodotto del numero di candidati di ogni cella vuota*** ci dà un'idea della dimensione dello spazio di soluzioni da esplorare per risolvere il problema. Ovviamente, man mano che i valori in ulteriori celle vengono fissati, la dimensione dello spazio delle soluzioni da esplorare diminuisce, fino a ridursi a 1.

Input e output. Il programma implementato deve ricevere in input un'istanza di Sudoku, memorizzata in un file di testo di 9 righe contenente, per ciascuna riga, 9 simboli scelti in $\{.\} \cup [1, 9]$. Il simbolo "." denota che la corrispondente cella è vuota. Un possibile input, ad esempio, potrebbe essere il seguente:

```
53..7....
6..195...
.98....6.
8...6...3
4..8.3..1
7...2...6
.6....28.
...419..5
....8..79
```

Il nome del file di input (eventualmente preceduto dal path di accesso) deve essere l'unico argomento ricevuto dal programma. Il programma deve stampare in output:

- la dimensione dello spazio delle soluzioni;
- il fattore di riempimento della matrice iniziale (ovvero la percentuale di celle non vuote dell'istanza di input);
- il numero di possibili soluzioni legali.

Implementazione. Oltre ad un algoritmo parallelo basato sul ***paradigma fork-join***, implementate una soluzione sequenziale da utilizzare per calcolare lo speedup del programma parallelo. In entrambi i casi, vi consiglio di non usare un approccio a forza bruta, verificando che i vincoli siano soddisfatti solo quando la scacchiera è completa, ma di verificare "strada facendo" la legalità delle soluzioni parziali ottenute, scartando fin da subito quelle inammissibili.

Esperimenti. A corredo dell'implementazione, eseguite una serie di esperimenti e misurate il tempo di esecuzione richiesto da ciascuno di essi, confrontando i tempi richiesti con quelli dell'esecuzione sequenziale e valutando lo speedup ottenuto. ***Discutete ed interpretate*** i risultati sperimentali ottenuti:

- Lo speedup è sempre maggiore di 1? Perché?
- Quali istanze richiedono più tempo?
- Esiste una correlazione tra fattore di riempimento, spazio delle soluzioni e tempo di esecuzione?

Benchmark. A breve saranno a disposizione sul sito Web del corso una serie di benchmark su cui testare il codice.

Documentazione. A corredo dell'implementazione (codice Java), inviatemi una relazione ***in formato pdf*** (max 5 pagine) che illustri:

- l'approccio algoritmico utilizzato e il modo in cui avete effettuato la parallelizzazione;
- le caratteristiche salienti dell'implementazione realizzata e le relative ottimizzazioni;
- le caratteristiche della/e piattaforma/e su cui avete eseguito l'analisi sperimentale;
- i risultati sperimentali ottenuti, tramite ***tabelle*** o ***grafici*** opportunamente commentati;
- la modalità per compilare ed eseguire il programma (da linea di comando).

Modalità di svolgimento: potete lavorare in gruppi composti da al più 3 persone.

Deadline: 2 gennaio 2016.