Trabalho 2 – Verificador de Sudoku Concorrente em Python

INE5410 – Programação Concorrente – UFSC
Profs. Márcio Castro e Giovani Gracioli

1 Introdução

O sudoku é um quebra-cabeça baseado na colocação lógica de números criado por *Howard Garns*, um projetista e arquiteto de 74 anos aposentado. O objetivo do jogo é a colocação de números de 1 a 9 em cada uma das células vazias numa grade de 9x9, constituída por 3x3 subgrades chamadas **regiões**. Considere que as linhas, colunas e regiões da grade são numeradas de 1 à 9. As regiões são numeradas da seguinte forma: região 1 (linhas e colunas de 1 à 3), região 2 (linhas de 1 à 3 e colunas de 4 à 6), região 3 (linhas de 1 à 3 e colunas de 7 à 9), região 4 (linhas de 4 à 6 e colunas de 1 à 3), região 5 (linhas de 4 à 6 e colunas de 4 à 6), região 5 (linhas de 4 à 6 e colunas de 7 à 9), etc. A Figura 1 apresenta um exemplo de um quebra-cabeça sudoku a ser resolvido.

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**

Figura 1: Exemplo de um quebra-cabeça sudoku.

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**

Figura 2: Exemplo de quebra-cabeça sudoku resolvido.

A regra para a colocação dos números nas células vazias é a seguinte. Em cada **coluna**, **linha** e **região** da grade, os números de 1 à 9 só podem aparecer uma única vez. Em outras palavras, não é permitido a repetição de um número em uma mesma linha, coluna ou região da grade. A Figura 2 apresenta um exemplo de uma solução para o quebra-cabeças da Figura 1. Note que, nesse caso, não há repetição de um mesmo número em uma mesma linha, coluna ou região da grade. Logo, essa solução está **correta**.

2 Definição do Trabalho

O trabalho 2 consiste em desenvolver um **validador de soluções de quebra-cabeças sudoku em Python de maneira paralela**. As soluções a serem validadas serão fornecidas através de um **arquivo texto**, o qual conterá um conjunto de **grades de tamanho 9x9**, separadas entre si por uma **linha em branco**.

Visando aumentar o desempenho do verificador, o seu programa deverá permitir que diferentes processos colaborem na correção das grades. Portanto, o **seu programa deve receber como parâmetros de entrada: (i) o nome do arquivo com as soluções a serem validadas, (ii) o número de *processos trabalhadores* e (iii) o número de *threads de correção* a serem utilizadas por cada *processo trabalhador***.

Após serem criados, os *processos trabalhadores* deverão **dividir o trabalho de validação das soluções** de sudoku fornecidas no arquivo. Cada *processo trabalhador* contará com um conjunto de *threads de correção* para verificar possíveis erros em cada grade destinada ao *processo trabalhador*. Portanto, a verificação das regras do jogo sobre as linhas, colunas e regiões para uma grade deverá, necessariamente, ser feita de forma concorrente por diferentes *threads de correção* do *processo trabalhador*.

A forma de divisão do trabalho computacional a ser realizado para validar todas as soluções fornecidas no arquivo deverá ser definida pelo grupo. Porém, deseja-se evitar ao máximo que *processos trabalhadores* e *threads de correção* sejam criados e permaneçam ociosos sem realizar nenhum trabalho. A solução deverá funcionar para diferentes números de *processos trabalhadores* e *threads de correção*, evitando-se, porém, a criação de *processos trabalhadores* e/ou *threads de correção* quando não for possível e/ou necessário.

Antes de começar o processamento de um quebra-cabeças, o *processo trabalhador* deve imprimir na tela `Processo P: resolvendo quebra-cabeças S`, onde `P` é um identificador único de *processo trabalhador* e `S` é um identificador único de quebra-cabeças, conforme ordem disposta no arquivo de entrada. Uma vez terminado o processamento do

quebra-cabeças pelo processo, o mesmo deverá imprimir na tela a quantidade de erros encontrados e, havendo erros, os locais em que as *threads de correção* encontraram os erros utilizando as seguintes siglas: `L` (erro em uma linha), `C` (erro em uma coluna) e `R` (erro em uma região). Além da sigla, deverá ser informado um número, que representará a linha/coluna/região em que o erro foi encontrado. O identificador de *processos trabalhadores*, *threads de correção* e quebra-cabeças deverá ser um **número inteiro sequencial** entre 1 e n , onde n é o número total processos, o número total de *threads de correção* de um *processo trabalhador* ou o número total de quebra-cabeças fornecido no arquivo de entrada. Em caso de erros encontrados, a ordem de apresentação seguirá a ordem crescente da numeração das *threads corretoras* (`T1`, `T2`, ...). A impressão da localização dos erros encontrados pelas *threads corretoras* deverá obedecer a seguinte ordem: primeiramente os erros encontrados nas linhas (`L`), depois colunas (`C`) e por fim nas regiões (`R`). O caractere "`;`" deverá ser utilizado para separar as informações dos erros encontrados por cada thread.

A Figura 3 mostra um exemplo de saída válida considerando a execução com um arquivo de entrada contendo apenas 2 soluções (uma correta e outra incorreta), 2 *processos trabalhadores* e 2 *threads de correção*. Na primeira solução (quebra-cabeças 1), nenhuma *thread de correção* encontrou erro. Por outro lado, a segunda solução (quebra-cabeças 2) continha erros. Neste caso, a thread `T1` encontrou um erro na coluna `C1`, na coluna `C9` e na região `R9`; a thread `T2` encontrou erros na linha `L4` e na coluna `C2`.

```
$ python sudoku solucoes.txt 2 2

Processo 1: resolvendo quebra-cabeças 1
Processo 2: resolvendo quebra-cabeças 2
Processo 1: 0 erros encontrados
Processo 2: 5 erros encontrados (T1: C1, C9, R9; T2: L4, C2)
```

Figura 3: Um exemplo de saída com 2 *processos trabalhadores* e 2 *threads de correção* em cada *processo trabalhador*.

A sua solução deverá seguir **rigorosamente** esse formato de saída. Logicamente, a ordem de apresentação dos resultados assim como a distribuição das tarefas entre *processos trabalhadores* e *threads de correção* poderá mudar em função da forma de divisão do trabalho adotada e também de uma execução para outra por se tratar de um programa concorrente.

Além da solução, você deverá entregar gráficos de *speedup* variando o número de processos e threads da solução, assim como uma pequena análise (relatório) sobre os resultados obtidos.

3 Grupos, Avaliação e Entrega

O trabalho deverá ser realizado em grupos de **3 alunos**. Os alunos serão responsáveis por formar os grupos com auxilio da ferramenta "**Escolha de Grupos - Trabalho 2 (T2)**" disponível no Moodle.

Pelo menos um dos integrantes de cada grupo deverá submeter um arquivo contendo o código fonte em python contendo a solução do trabalho através do Moodle no prazo estipulado. **Trabalhos não entregues no prazo receberão nota zero.**

Após a data limite para entrega, os alunos deverão apresentar o trabalho ao professor assim como mostrar sua solução em funcionamento. As apresentações serão feitas durante as aulas conforme o cronograma da disciplina disponível no Moodle.

O professor irá avaliar não somente a corretude mas também o desempenho e a clareza da solução. Durante a apresentação, o professor irá avaliar o **conhecimento individual dos alunos sobre os conteúdos teóricos e práticos vistos em aula e sobre a solução adotada no trabalho**. A nota atribuída à cada aluno i no trabalho ($NotaTrabalho_i$) será calculada da seguinte forma, onde A_i é a nota referente à apresentação do aluno i e S é a nota atribuída à solução do trabalho:

$$NotaTrabalho_i = \frac{A_i * S}{10} \tag{1}$$

Como indicado pela fórmula mostrada acima, **a nota atribuída à solução adotada será ponderada pelo desempenho do aluno durante a apresentação do trabalho**. Por exemplo, se o professor atribuir nota 10 para a solução adotada pelo grupo mas o aluno receber nota 5 pela apresentação – devido ao desconhecimento dos conteúdos teóricos, práticos e/ou da solução do trabalho – a sua nota final do trabalho será 5. A ausência no dia da apresentação ou recusa de realização da apresentação do trabalho implicará em nota zero na apresentação, fazendo com que a nota atribuída ao aluno também seja zero.