Trabalho 2 – Verificador de Sudoku Concorrente em Python

INE5410 – Programação Concorrente – UFSC
Profs. Márcio Castro, Odorico Mendizabal

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 concorrente**. 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 resolve 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. Quando uma *thread de correção* encontrar um

erro na solução, ela deverá imprimir na tela `Thread T: erro na AREA X`, onde `T` representa um identificador único de *thread de correção* de um *processo trabalhador*, `AREA` indica se foi em uma `linha`, `coluna` ou `região` e `X` representa o número da `linha`, `coluna` ou `região` onde o erro se encontra. 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. Ao final da validação de cada quebra-cabeças, o *processo trabalhador* deverá imprimir `Erros encontrados: E`, onde `E` é o número total de erros encontrados na solução que acabou de ser validada pelas *threads de correção* do *processo trabalhador*.

```
$ ./sudoku solucoes.txt 2 2

Processo 1 resolve quebra-cabeças 1:
Erros encontrados: 0.

Processo 2 resolve quebra-cabeças 2.
Thread 1: erro na coluna 1.
Thread 2: erro na coluna 2.
Thread 1: erro na coluna 9.
Thread 2: erro na linha 4.
Thread 1: erro na regiao 6.
Erros encontrados: 5.
```

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

A Figura 3 mostra um exemplo de saída 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, nenhuma *thread de correção* encontrou erro. Por outro lado, a segunda solução continha erros. A sua solução concorrente 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.

3 Grupos, Avaliação e Entrega

O trabalho deverá ser realizado **em duplas**. A escolha dos grupos, o desenvolvimento e a entrega do trabalho deverão ser feitos via **Github Classroom**. Um pequeno guia sobre o Github Classroom e o Git foi disponibilizado no Moodle. Leia atentamente o guia antes de fazer a escolha dos grupos e começar a atividade. **ATENÇÃO**: Serão permitidos somente 25 grupos! Defina sua dupla conversando com seus colegas e, somente após a definição da dupla, crie um grupo no Github Classroom. Foi disponibilizado no repositório base um exemplo de arquivo de entrada contendo 2 soluções (uma correta e outra incorreta) para realização de testes iniciais.

Será considerado entregue a versão do repositório criado via Github Classroom com o último *commit* realizado até a data limite para submissão: **13 de Dezembro às 23:59**. Na versão entregue deve constar além da implementação, um **documento no formato pdf de no máximo 3 páginas respondendo as seguintes perguntas**:

1. Qual foi a lógica de divisão de tarefas entre *processos trabalhadores* e *threads de correção*? Justifique sua resposta.
2. Quais são os limites dos números de *processos trabalhadores* e *threads de correção* da sua solução? Por quê?
3. Quais métodos de sincronização foram utilizados na solução proposta? Justifique a necessidade deles.
4. Você obteve ganho de desempenho com a solução paralela? Indique a configuração do computador no qual foram realizados os experimentos, assim como o ganho de desempenho obtido.