Project Sudoku

Uma breve história do Código

Bruno Netto

July 21, 2018

Dedicated to all project eulers out there.

Contents

List of Figures

List of Tables

Prefácio

Esse é um relatório de um projeto inspirado no problema 96 do Project Euler. (`https://projecteuler.net/problem=96`)

Project Euler - O Site

Project Euler é um site com problemas de matemática e computação incriveis e tão viciantes quanto divertidos. Visto que além tentarmos resolver em menos de 1 minuto utilizando de boas práticas computacionais como programação dinâmica ou até mesmo força-bruta, são bons problemas para se compartilhar com os colegas e professores. É um site super recomendado para desenvolver e aperfeiçoar técnicas de programação em qualquer linguagem que se deseja aprender.

Project Euler - O Curso

Project Euler foi um curso ministrado pelo professor Bernardo Freitas do IM-UFRJ. O curso foi focado em alguns problemas do site project euler com o intuito de aprendermos técnicas de problem solving tais como backtracking, programação dinâmica além de como manipular certas estruturas de dados, percorrendo árvores

Estrutura do texto

O primeiro capítulo é uma curta introdução ao problema e a linguagem utilizada. O segundo e terceiro capítulo são comentarios em cima do código tanto do solver quanto do gerador de sudokus. O último capítulo são dados estatísticos principalmente de performance.

Página com o Código

O site[1] deste arquivo contém:

[1] `https://github.com/Aristt27/Project-Sudoku`

- Um link para (baixar gratuitamente) a última versão desse documento.
- O Código utilizado e comentado por este.

Acknowledgements

- Link[2] to download LaTeX source for this document can be found here.
- A special word of thanks goes to Professor Don Knuth[3] (for TEX) and Leslie Lamport[4] (for LATEX).
- I'll also like to thank Gummi[5] developers and LaTeXila[6] development team for their awesome LATEX editors.
- I'm deeply indebted my parents, colleagues, friends and teachers for their support and encouragement.

[2] https://github.com/amberj/latex-book-template
[3] http://www-cs-faculty.stanford.edu/~uno/
[4] http://www.lamport.org/
[5] http://gummi.midnightcoding.org/
[6] http://projects.gnome.org/latexila/

1

Introdução

“ $\frac{d^i f}{dx^i}$ *(...) sequelou.”*

– -Xavier, Pedro

1.1 O Problema

O foco do projeto é tanto resolver quanto criar sudokus com unicidade de solução. Um sudoku é um tabuleiro 9x9 preenchido parcialmente com dígitos de 1 a 9, de maneira que cada dígito só aparece uma vez por linha, coluna ou sub-quadrado 3x3.
Estes podem ser resolvidos através de lógica e deduções feitas a partir dos números que já aparecem e suas posições. Entretanto, em alguns casos, é necessário um recurso mais poderoso: a memória. Que é a maior vantagem do computador para resolver este problema através de backtrackign e tentativa e erro.

1.2 A Linguagem

Todo código comentado nesse trabalho será de Pyhton 3. A maior das razões é pela interface e qualidade do Jupyter Notebook e sua facilidade de alterar e identificar tanto entradas quanto saídas de código. Outra razão é que usamos e abusamos da estrutura de dados lista, que é um recurso trivialmente implementado e acessado em python.

1.3 O Tabuleiro

O tabuleiro tanto de entrada quanto de saida de cada código, tanto solver quanto gerador, consiste numa lista com 9 listas dentro, e cada uma dessas 9 listas contem 9 números de 0 a 9. De maneira que os 0 representam

os quadrados não preenchidos do sudoku, e os numeros de 1 a 9 são eles próprios. Foi feita essa escolha pois fica facil de vizualizar o jogo enquanto é resolvido, e fácil de programar tambem.

1.4 O Código 0

1.4.1 Sudoku_reader.py

Nessa seção explicarei o código contido no arquivo sudoku_reader.py, responsável por acessar um arquivo contendo uma string com varíos sudokus e retornar uma lista com listas contendo os tabuleiros formatados para o solver. Na primeira função eu removo a string "Grid XX", acessando um arquivo txt e coloco as linhas, numa string só, em listas.

Já na segunda função, remove-se os "\n" e fragmenta uma string dentro de uma lista transformando os numeros em inteiros. A terceira e ultima função é um map da segunda função na primeira, aplicando ela em todos as listas dentro da lista retornada pela primeira função. Resultando numa lista de listas onde cada lista é um sudoku.

2

O Solver

"In order to understand recursion, one must first understand recursion."

– -Unknown

2.1 O Objetivo

Com essa parte do código nosso objetivo é dado um sudoku no formato de lista de listas, com numeros de 0 a 9 retornar sua solução (apenas numeros de 1 a 9 sem repetiçoes em suas linhas colunas e sub-quadrados); Portanto, nada mais justo que ensinar pro programa a perceber quando um sudoku está resolvido que, no nosso caso, é quando não tem mais zeros se não forem feitos movimentos errados.

2.2 As ideias para o Solver

A primeira ideia é saber quais são os movimentos possíveis e válidos[1] em um dado sudoku. Ou seja, obter uma lista, para cada 0 em um tabuleiro, com todos os possiveis movimentos. Com isso, já temos um método força bruta para resolver o problema, tentar todas as combinações de movimentos e, eventualmente (em uns 10 anos), chegar na solução desejada utilizando o mínimo de memória possivel.

2.2.1 Ordenando os movimentos

Podemos acelerar esse algoritmo básico dando uma ordem aos movimentos, visto que podemos ordenar as casas do tabuleiro pela quantidade de alternativas possiveis em cada uma delas. Note que se uma casa tem apenas um

[1] Sem repetir nenhum numero em linhas,colunas e sub-quadrados.

movimento possível, então este movimento caminha para a solução. E assim, podemos renovar a lista dos possíveis movimentos e repetir o algoritmo sempre procurando as casas com menos alternativas.

Note que se em alguma casa só tiver 1 alternativa, ou seja, todos os outros 8 numeros ja foram colocados na mesma linha, coluna e sub-quadrado desta casa, então esse movimento é válido e trivial. Portanto, iremos focar em achar essas casas sempre, antes de começar a tentar advinhar.

2.2.2 Usando um pouco mais de memória

De fato, podemos ir guardando na memória, a cada movimento, as mudanças feitas no tabuleiro, e se chegarmos em um tabuleiro impossivel, ou seja, que não possa mais progredir mas ainda tenha zeros, sejamos capazes de voltar atrás e realizar um movimento diferente. Essa é a ideia principal do backtracking. A cada movimento em um quadrado que tenha mais de 1 alternativa, salvamos o tabuleiro, guardando uma cópia com um dele na memória;f para quando alterarmos o primeiro não modificarmos a cópia salva e assim poder voltar atrás caso chegarmos em um sudoku inválido[2].

2.2.3 Recursividade, Recursividade, Recursividade

Estamos caminhando para um algoritmo recursivo que verifica os movimentos possiveis, ordena pela quantidade de alternativas, salva o tabuleiro se for escolher entre algum movimento e repete enquanto for possível. Caso não seja, voltamos para o último tabuleiro salvo, e tomamos um galho diferente da árvore, ou seja, fazemos jogadas diferentes para nao entrarmos em um círculo vicioso. Repetindo até acabarem os zeros do tabuleiro.
Mas foi necessário um cuidado especial devido à profundidade máxima de recursão do Jupyter Notebook, visto que a função se chamava a cada movimento, errado ou não, por isso foi necessário alterar para um loop while e separar as funções entre um pré-processamento

2.2.4 Fater Sudoku

A ideia que mais acelerou o código foi uma outra maneira de determinar um movimento certo; Até então eu procurava os quadrados com apenas uma jogada possível, entretanto se nas possibilidades de alguma linha, coluna ou sub-quadrado, houver algum numero que só apareça 1 vez, sozinho ou não, então esse movimento tambem é certo. Mas a maior dificuldade foi encontrar em que lugar eu coloco essa nova verificação sem perder muito tempo verificando.

[2] Que ainda tenha zeros mas não tenha mais movimentos válidos

2.2.5 X-wings e Millennium Falcons

A ultima ideia para o código, que não foi implementada, são as estratégias que dão mais prioridade para as posições dos números. Criando alguns links que são mais fáceis para o olho humano ver que para um computador. Mas devido a intensa recursividade é razoavel supor que não seja um grande ganho de velocidade, visto que qualquer processo de verificação seria rodado algumas centenas de vezes, gerando uma perda de tempo tambem.
A partir dessas técnicas e limitando a memória utilizada se é capaz de criar um solver mais 'humano', e, limitando o número de tecnicas podemos construir diversos níveis de habilidade do solver.

2.3 O Código I - A ameaça fantasma

2.3.1 Solver_funcs.py

No arquivo solver_funcs.py onde estão localizadas as funçoes chamadas pelo solver.py, começo definindo funções para o pré-processamento do sudoku. Funções que contam o numero de zeros, ou seja, os espaços livres, função que cria uma lista com todos os movimentos possiveis para cada casa do sudoku, e uma que verifica quais desses não são possiveis; gerando assim a chamada guess_list, uma lista com listas representando as possiveis jogadas.
Logo mais, já com a ideia de ordenar as jogadas, defino a next_move e next_branch, a primeira procura na guess_list a casa com menos alternativas. A segunda escolhe uma das alternativas, se forem mais de uma, de maneira que não escolhe a mesma duas vezes no mesmo galho (branch). Este é uma lista que guarda os caminhos que falharam e resultaram em um sudoku sem solução. A partir dele sabemos por onde percorrer na árvore das possibilidades.
Em seguida, para não chamar muitas vezes a função do pré-processamento que vê quais são os movimentos possiveis a cada jogada realizada, defino a função que atualiza essa lista com base no local e do valor da jogada. E finalizo definindo as funções para verificar se algum numero aparece uma só vez nas possibilidades de cada sub grupo do sudoku (linhas, colunas ou sub-quadrados).

2.3.2 Solver.py

Já neste outro arquivo, estão separadas as duas funções que recebem o tabuleiro do sudoku; a que faz um pré-processamento, sudoku_starter, criando as variáveis como a lista para backups de tabuleiros, a lista que endereça o galho atual e numero de movimentos restantes. Alem disso, nessa mesma função se realiza os movimentos mais 'triviais', pois se localizam nos quadrad-

inhos do sudoku que só tem uma alternativa para movimento, ou aparece apenas uma vez nas possibilidades de cada sub grupo do sudoku.

Já a outra função, recebe todas essas variáveis e utiliza a next_branch para chutar nos quadrados com menos alternativas ,quando for necessário, e se chama recursivamente até acabarem os movimentos possiveis. E das duas umas; ou parou numa solução (sem zeros no tabuleiro). Ou então chegamos no ponto que voltamos para o tabuleiro salvo no backup e percorremos outro galho da árvore das possibilidades.

3

O Gerador

"É na sola da bota, é na palma da bota"

– -Lispector, Analu

3.1 O Objetivo

O objetivo dessa parte do código é gerar sudokus com unicidade de solução e o minimo de dicas rapidamente.

3.2 As ideias para o Gerador

A principal ideia foi criar um teste para a unicidade de solução e depois ir removendo numeros aleatoriamente enquanto o sudoku não encorporar múltiplas soluções.

3.2.1 A Unicidade e o método de caminhos opostos

Pela maneira que o solver foi criado, tem um teste muito simples para a unicidade do sudoku que envolve a função que determina um chute entre 2 ou mais alternativas de um quadrado.

Se sempre fizermos as escolhas dentre as alternativas em uma determinada ordem, suponha que da esquerda para a direita, e encontrarmos uma solução. E depois fizermos as escolhas na ordem contrária, ou seja, da direita para a esquerda, e encontrarmos uma solução, se elas forem iguais então o sudoku tem solução unica. Visto que estamos percorrendo a árvore das alternativas por dois caminhos opostos, dando o nome ao método.

3.2.2 Removendo as dicas e minimos locais

Com esse teste, podemos criar uma função que tenta remover 64 numeros aleatórios, a partir de um sudoku ja resolvido,visto que o menor numero de

dicas possiveis que mantem a unicidade de solução é 17. Eu guardei em um mini banco de dados 3 sudokus e suas soluções para serem seeds geradores de sudokus; um com 17 dicas, um que é a solução mais trivial de um sudoku em branco, e um no meio dos dois.
Construimos a função de maneira que ela guarde os minimos locais e possamos regular a quantidade máxima iterações dela visto que a mesma pode demorar alguns segundos a cada 10 iterações.

3.3 Ideias Finais

Apartir das técnicas 'humanas' como x-wing ou swordfish, podemos criar sudokus mais complexos procurando pelas posições dessas estrategias e tentando evitá-las removendo quadrados específicos impedindo o uso dessas técnicas.

3.4 O Código II - Ataque dos Clones

3.4.1 Generator.py

Neste arquivo começo criando variaveis para armazenar alguns exemplos de sudoku bem diferentes, para servirem de ponto de partida para criação dos outros. Logo em seguida defino a função unique_sudoku que realiza o teste de unicidade de solução pelo método de caminhos opostos mencionado anteriormente sendo que dentro dela eu defino a função next_branch duas vezes visto que ela que controla o caminho dos chutes. Alem de testar a unicidade ela tambem retorna o tempo que cada caminho levou para resolver o sudoku, o que se torna um dado bacana para comparar entre os diferentes sudokus.
Concluo o arquivo criando as funções que são responsáveis por gerar os sudokus a partir das soluções já definidas. Removendo, 1 a 1, numeros aleatoriamente e testando sua unicidade, guardando os minimos locais e começando novamente até chegar ou a 17 dicas, ou, no limite de iterações.

4

Estatísticas

"Cachorro-quente é sanduiche"

– -Luiz, Vitor

4.1 Sudoku Nulo

A figura 4.1 ilustra os galhos opostos entre a esquerda e a direita, e as consequencias de percorrer por eles enquanto o solver 'resolve' um sudoku em branco com 2 das suas soluções 'antípodas'.

4.2 Os 50 sudokus do Project euler

4.2.1 Tempo de Leitura e Formatação

Resultado de um % timeit na função [sudoku_starter(puzzles('p096_sudoku.txt')[i]) for i in range(50)]: 1 loop, best of 3: 2.68 s per loop

4.2.2 Comparação entre os galhos opostos

Comparção entre sempre chutar pela esquerda e sempre chutar pela direita nos 50 sudokus e os sudokus com maior amplitude;

Table 4.1: Tempo de Exec dos 50 sudokus do PE

sudokus	Solv. Esquerda	Solv. Direita
Sudoku 30	0.82770s	0.27144s
Sudoku 46	0.29219s	0.90378s
Sudoku 49	1.22874s	0.11900s
tempo total(50)	3.89734s	3.19146s

```
In [151]: null_sudoku

Out[151]: [[0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

```
In [164]: %time sudoku_starter(null_sudoku) #esquerda

          Wall time: 11 ms

Out[164]: array([[1, 2, 3, 4, 5, 6, 7, 8, 9],
                 [4, 5, 6, 7, 8, 9, 1, 2, 3],
                 [7, 8, 9, 1, 2, 3, 4, 5, 6],
                 [2, 3, 1, 6, 7, 4, 8, 9, 5],
                 [8, 7, 5, 9, 1, 2, 3, 6, 4],
                 [6, 9, 4, 5, 3, 8, 2, 1, 7],
                 [3, 1, 7, 2, 6, 5, 9, 4, 8],
                 [5, 4, 2, 8, 9, 7, 6, 3, 1],
                 [9, 6, 8, 3, 4, 1, 5, 7, 2]])
```

```
In [157]: %time sudoku_starter(null_sudoku) #direita

          Wall time: 6.99 ms

Out[157]: array([[9, 8, 7, 6, 5, 4, 3, 2, 1],
                 [6, 5, 4, 3, 2, 1, 9, 8, 7],
                 [3, 2, 1, 9, 8, 7, 6, 5, 4],
                 [8, 7, 9, 4, 3, 6, 2, 1, 5],
                 [2, 3, 5, 1, 9, 8, 7, 4, 6],
                 [4, 1, 6, 5, 7, 2, 8, 9, 3],
                 [7, 9, 3, 8, 4, 5, 1, 6, 2],
                 [5, 6, 8, 2, 1, 3, 4, 7, 9],
                 [1, 4, 2, 7, 6, 9, 5, 3, 8]])
```

nulo.png

Figure 4.1: Uma 'solução' de um sudoku nulo (sem dicas)

4.2.3 Tempo de Exec do Gerador

Após rodar o gerador 50 vezes, com as 3 seeds diferentes, o que demorou cerca de 300 segundos para cada seed, o menor numero de dicas foi 28 para seed 0, 28 para seed 1 e 29 para a seed 2. O que demonstra que ele não consegue ser muito eficaz, visto que com apenas 5 segundos ele consegue gerar sudokus com 31 dicas.