UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

ALGORITMOS I - TP3

Breno Poggiali de Sousa
2018054800

November 24, 2019

Sumário

1 Sobre o Problema

O objetivo deste trabalho é desenvolver uma heurística que seja capaz de resolver sudokus de diferents tamanhos. Serão recebidos 3 valores: N, I, J. Houve uma pequena mudança da forma que eu fiz com a forma que foi solicitada no trabalho. N deveria se tratar do tamanho do sudoku (NxN) e isso foi feito! Porém, em meu trabalho, I se tratou das linhas e J das colunas. Isso foi feito pois seria mais intuitivo para outras pessoas entenderem o trabalho e, também, para mim.

No caso, I e J definem os tamanhos dos retângulos. I define o tamanho nas linhas e J o tamanho nas colunas. Ex: Um sudoku com N=9, I = 3 e J = 3, é um sudoku tracional. Não obstante, um sudoku com N=8, I = 4 e J = 2, é um sudoku de tamanho 8x8, com 8 quadrados de largura 2 e de comprimento 4.

A ideia principal é, a partir disso, tentarmos criar uma heurística que seja capaz de solucionar esses sudokus ”especiais”. No meu caso, tentarei criar uma heurística parecida com a forma que a nossa cabeça tenta solucionar um sudoku, sem que sejam necessarios algoritmos de backtracking que envolvem o ”chute” de valores.

2 Heurísticas adotadas

Breve explicação sobre minha heurística

A heurística escolhida por mim foi baseada em um sudoku normal e como ele é resolvido. No geral, ela é bem simples de entender. Declarei duas matrizes. Uma para colocar todas as possibilidades de dígitos em uma posição e outra que eu chamei de ”Checked”. Todas as duas matrizes são do tamanho NxN. Porém, a matriz normal (sudoku), em cada posição é possível colocar um array de todas as possibilidades naquela posição. Já na matriz checked, cada posição é um boolenano (true ou false). O que foi feito consistiu em ir iterando pela matriz e, quando chegasse no momento em que em uma posição só houvesse uma única opção para ela, marcavamos o checked dessa opção como checked e atualizariamos as possibilidades de todas as outras posições da matriz.

Dessa forma, desenvolvemos um algoritmo similar à resolução de sudokus tradicionais. Não obstante, criamos uma solução que nem sempre irá testar todas as opções e resolver todos os sudokus resolvíveis.

Porque optou-se por determinada heurística?

A heurística descrita acima foi optada, principalmente, por dois motivos. O primeiro deles e o mais importante, porque eu queria desenvolver um algoritmo que fosse criado diretamente da minha cabeça e que seria a forma mais próxima possível de como eu mesmo resolvo sudokus, sem que simplesmente fosse copiado da internet um algoritmo qualquer que utiliza de backtracking e que resolve todos os sudokus possíveis com solução.

O segundo motivo tem a ver com o que eu respondi no final da pergunta acima. Eu tentei criar um algoritmo mais próximo do possível de como eu mesmo resolvo sudokus. Desse forma, apesar de se tratar de um algoritmo simples, se trata de um algoritmo que resolve boa parte dos sudokus tradicionais.

3 Análise de complexidade

Tempo: O(N^6). Isso porque, apesar de para montar a matriz checked ser O(n^2) e de gastarmos para atualizar essa mesma matriz ser a mesma complexidade(O(1), n^2 vezes), onde é a parte mais custosa é no caso da matriz sudoku. Nessa matriz, atualizamos ela da seguinte forma: Iteramos cada posição da matriz N^2 vezes, de forma que passamos n^4 vezes por toda a matriz, após isso, no pior caso, atualizamos cada quadrado interno da matriz n^4 vezes.

Dessa forma, passamos N^4 vezes pelas posições da matriz e mais n^2 vezes em cada quadrado, de forma que temos uma complexidade de n^6. Essa n^2 vezes em cada quadrado acontece da seguinte forma: cada quadrado, tem no máximo n elementos, que possuem, no máximo, todos n elemntos em cada uma das suas posições, sendo asssim, n^2 vezes, que, junto com o n^4 da função principal, retorna para nós uma complexidade final de O(n^6).

É importante salientar aqui que, como nosso N máximo é 9, 9^6 = 531441. E, como sabemos que 531441 está dentro de 10^5 e sabemos que 10^5 roda tranquilamente dentro de 1 segundo, é muito tranquilo assumirmos essa abordagem dada as constraints do trabalho.

Espaço: O(N^3). Isso porque, o que é usado de espaço são duas matrizes do tamanho n*n. Uma delas, a checked, é do tamano N*N, com um booleano em cada posição, de forma que, no pior caso, ela ocupa 81 posições (já que sabemos que

nossas constraints não passam de N = 9). Já a segunda matriz, que é a mais custosa, em um pior caso (em uma matriz vazia), tem tamanho N*N e em cada posição, podem ser 9 possibilidades.

Dessa forma, o número de elementos guaradados são 81*9 = 9^3 = 729. Porém, como 729 é um número consideravelmente muito baixo e visto que temos garantia que iremos trabalhar, necessariamente, com sudokus nunca maiores que 9x9, é muito tranquilo utilzarmos esse valor de Big O, que, apesar de ser horrível em problemas maiores, nesse, é muito tranquio de ser aproveitado.

4 Avaliação Experimental

Tempo de execução dos algoritmos

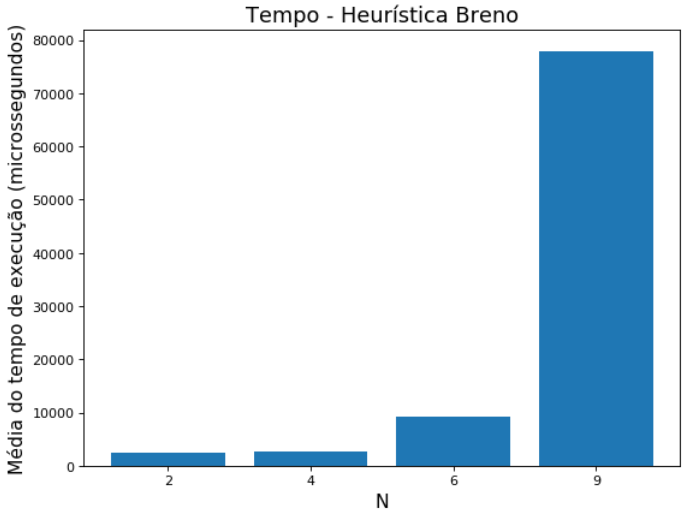
1. Tempo e quantidade de memória utilizada no pior caso:

Assim como foi explicado acima, no pior caso, temos um uso da quantidade de memória dentro de $O(N^3)$. Isso porque, a matriz que construimos que ocupa mais espaço, no **pior caso**, que seria com o **sudoku todo em branco**, ocupa NxN posições = 81 posições, onde, dentro de cada posição, existe um array de tamanho N. O que nos dá a complexidade de de $O(N^3)$.

Já quanto ao tempo, assim como também explicamos acima, temos $O(N^6)$ no pior caso. Isso porque, no **pior caso**, que também seria o **sudoku todo em branco**, essa matriz entra N^4 vezes em cada posição e em cada posição, vasculha todos os elementos presentes do quadrado (que podem ser até N^2) no caso da matriz em branco.

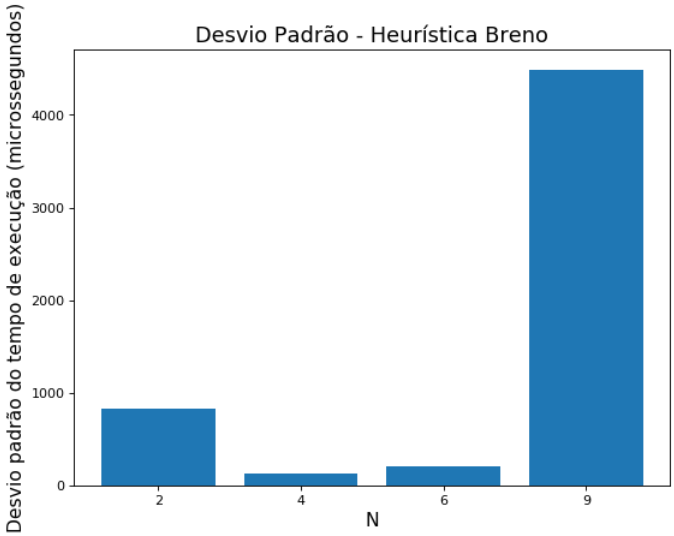
2. Média e Desvio padrão do tempo de execução do algoritmo:

Média:



Podemos perceber na análise da Média de Tempo de execução do algoritmo, que ele cresce MUITO rápido à medida que N cresce. E, assim como analisamos, esse muito rápido está bem próximo de N^6.

Desvio Padrão:



Já na análise do desvio padrão, podemos perceber que ele é consideravelmente maior para N = 9. Isso ocorre pois, como o tempo de execução com N = 9 é tão maior, é natural termos uma discrepância maior.

3. Quais os formatos de tabela do Sudoku a heurística obteve as melhores soluções:
Sobre a minha heurística, ela acaba obtendo as melhores soluções em 2 casos:

1. Os casos mais simples, que são com sudoku menores. Nesses casos, por ser muito pequeno, com poucos elementos ela já consegue decifrar facilmente a solução.

2. Os casos em que são sudokus tradicionais do tipo ”fáceis” e até do tipo ”médio”. Isso porque foram nesses casos que eu me baseei para montar a minha heurística. Nesses tipos de sudoku, é possível descobrir valores novos apenas com as operações mais elementares do sudoku, que são conferindo as linhas, as colunas, os quadrados, e se um determinado elemento só pode ser em uma posição do quadrado e nenhuma outra mais.

5 Conclusão

Sendo assim, foi muito interessante termos feito tal trabalho. Isso porque foi possível aprendermos a complexidade de um problema NP e quais são as nuâncias na prática. Conseguirmos praticar também, formas de inventar uma heurística para um determinado problema, o que pode agregar muito em nossa vida acadêmica e profissional.

Por fim, o trabalho foi muito interessante para praticarmos um pouco mais sobre sudoku. Tal algoritmo é de suma importância, visto que requer muito raciocínio lógico e já chegou inclusive, a ser questão do ENADE no ano de 2014.

Bibliografia

ALMEIDA, J. M. (2019) *Slides da disciplina de Algoritmos I*

ZIVIANI, N. (2006). *Projetos de Algoritmos com Implementações em PASCAL e C: Capítulo 9: Problemas NP-Completo e Heuristicas*. Editora Cengage.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. (2009) *Introduction to Algorithms*. MIT Press.