Trabalho Prático 3 - Algoritmos 1

Tales Panoutsos Malheiros Lima

Novembro 2019

1 Descrição do Problema

Para esse trabalho foi passado que se deveria implementar uma um programa, que encontrasse uma solução para pelo menos a maior parte dos Sudokus próprios, isto é, que possuem apenas uma solução. A estratégia adotada foi o backtracking, ou seja, dada uma instância do problema, o algoritmo implementado testa todas as escolhas válidas de números para cada posição, até que encontre uma solução. Porém, para tornar o algoritmo mais inteligente, a escolha de qual posição preencher primeiro não é feita aleatoriamente. O processo de escolha de posição vai ser melhor descrito abaixo.

2 Algoritmo

O algoritmo implementado é bem intuitivo, e é muito semelhante à maneira com que os seres humanos costumam jogar Sudoku.

Tudo o que o algoritmo faz é varrer todas as posições em branco e procurar a que possui menos possibilidade de preenchimento, isto é, a que está restrita a menos números. Feito isso, para cada número válido naquela posição, o algoritmo a preenche e se chama recursivamente.

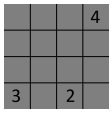
Assim que a tabela for totalmente preenchida, o algoritmo retorna essa nova tabela.

2.1 Exemplo:

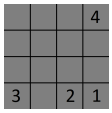
Suponha que um Sudoku quatro por quatro com quadrantes dois por dois e com a seguinte configuração seja entregue ao algoritmo:

3 Análise de complexidade

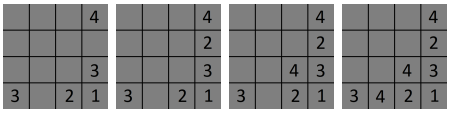
Como a recursão foi usada para a implementação desse solução, podemos olhar para a árvore de recursão formada para analisar a complexidade de tempo e espaço do algoritmo.



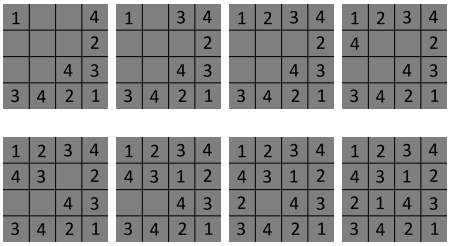
O algoritmo, calculando o número de possibilidades de números para cada posição da matriz, chega a conclusão que a posição com menor número de possibilidades é a posição A[3,3]. Nessa posição não se pode ser posto nem um dos números: dois, três ou quatro. Sendo assim o algoritmo o completa com o único número restante, o um.



Após colocar um na posição A[3,3], o algoritmo procura outra posição com apenas uma possibilidade, e continua a preencher analogamente.



Em determinadas situações com a seguinte, não há nenhuma posição com apenas uma escolha de número possível. Quando isso acontece, o que o algoritmo faz é: acha a posição com menor número de escolhas válidas e, para todas as escolhas válidas para essa posição, chame preencha a matriz com essa tal escolha, e chame a função recursivamente. Ou seja, quando houver mais de uma escolha, teste todas em ordem crescente. Por sorte, a primeira escolha já nos levou a uma solução dessa instância.



A árvore é formada da seguinte forma. O vértice inicial da árvore corresponde a configuração entregue para o algoritmo, e seus filhos correspondem as tentativas que o algoritmo fará. Logo, as folhas dessa árvore, serão configurações onde todas as posições já não possuem nenhuma escolha válida. Ou seja, algumas das folhas(as mais baixas) serão soluções, outras, serão "becos sem saída", onde já não se pode preencher nenhuma posição sem desrespeitar as regras, mesmo que haja posições em branco.

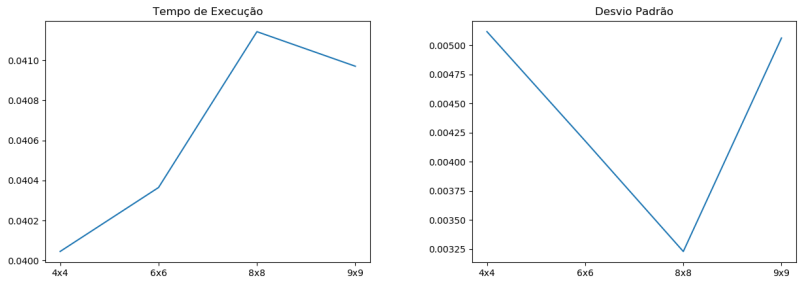
Portanto, como a estratégia de recursão é utilizada, a complexidade de espaço depende da pilha formada, que é correspondente a altura da árvore de recursão. Essa pilha, no pior dos casos, guarda uma matriz para cada chamada recursiva. Logo, como cada chamada recursiva corresponde ao preenchimento de uma posição, no pior dos casos é necessário que se guarde n^2 matrizes de tamanho, n^2, por tanto a complexidade de espaço é $O(n^4)$.

Para complexidade de tempo, temos que no pior dos casos, seria o número de vértices na árvore. Como cada vértice corresponde a uma determinada tentativa, o pior dos casos é quando é necessário testar todas as escolhas possíveis até que se acerte a solução. Isso nos dá uma complexidade exponencial $O(n^{2n})$, o que é esperado de uma estratégia de backtracking.

4 Avaliação experimental

Para a avaliação experimental desse algoritmo foram usados apenas sudokus próprios, isto é, que possuem solução única, portanto, já que o algoritmo para, assim que encontra a solução, veremos que ele não roda por tanto tempo quanto esperado considerando a sua complexidade.

Na análise dos tempos, para evitar conclusões erradas graças a má execuções cada sudoku foi resolvido 100 vezes, e para cada tamanho em 4x4, 6x6, 8x8 e 9x9, foram resolvidos 5 sudokus diferentes, ou seja, para cada tamanho foram executados 500 sudokus. Retirando a média e o desvio padrão para cada tamanho, chegamos ao seguinte resultado:



Podemos perceber claramente que o algoritmo tem mais dificuldade com os sudokus de tamanho 8x8. Isso pode ser explicado pelo fato de que, como o algoritmo usa a estratégia backtracking, ao chegar em uma resposta errada, ele volta e tenta de outra forma. Logo, provavelmente os sudokus 6x6 usados para o teste são sudokus em que o algoritmo não acertou de primeira, o número que deveria colocar em alguma situação em que havia mais de uma escolha válida.

Além disso, como era o esperado, o caso em que ele tem menor tempo de execução é o de instâncias de tamanho 4x4.

5 Conclusão

O algoritmo, assumindo que exista, sempre acha uma resposta correta, porém, como se pode ver na análise complexidade, ele pode demorar muito quando o a instância é muito grande, logo, ele não deve ser usado caso se necessite de um tempo útil para grandes instâncias.