UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA (DAINF)
CURSO ENGENHARIA DE COMPUTAÇÃO

ANNA CAROLINE DE OLIVEIRA SOUSA, GUILHERME ARISTIDES MARCOS

SUDOKU COM BACKTRACKING

TRABALHO PARA DISCIPLINA ESTRUTURA DE DADOS 1

Curitiba, 2021

Resumo – O presente artigo possui como objetivo a descrição do problema selecionado pelos discentes – Sudoku com Backtracking – bem como a sua resolução, descrição das escolhas seguidas e seus principais desafios encontrados durante a execução do mesmo. Não obstante, é notória a contribuição do aprendizado bem como para nossa formação os conhecimentos adquiridos e aprimorados com o desenvolvimento do mesmo.

Palavras-chave: sudoku, recursividade, pilha.

Descrição do problema e sua aplicação

Sudoku, proveniente do acrônimo da expressão "Os números devem ser únicos" é um jogo baseado na colocação lógica de números, com a finalidade de preencher as posições vazias. Outrossim, durante todo o jogo é necessário a análise de quais números são possíveis – pois, o mesmo não permite a repetição de números em suas colunas, linhas e submatrizes – além de que, é imprescindível ressaltar que o mesmo já apresenta algumas posições previamente preenchidas e que devem ser consideradas para o preenchimento das demais.

Concomitantemente, tendo isto em vista é possível averiguar que o jogo Sudoku é um jogo lógico e racional. Pois, para a resolução do mesmo a cada nova posição acessada é necessária uma série de análises, com o intuito de tomar decisões que cumpram os requisitos bem como suas regras – dentre as possibilidades apresentadas – de modo a encontrar a melhor solução para completar todas as posições existentes. Não obstante, é notório que cada posição ocupada no Sudoku implica em novo espaço amostral para as demais posições que estão próximo ao mesmo, condicionando em novas buscas e decisões a serem tomadas.

Por fim, é factível que para o desenvolvimento bem como a resolução do jogo é necessário ter muita atenção sobre os valores faltantes e presentes no tabuleiro. Apesar, de haver métodos e dicas – como a varredura, análise e emparelhamento – que possuem o objetivo de auxiliar na resolução do Sudoku, no presente trabalho gostaríamos de propor e discutir um método diferente dos tradicionais, onde através de funções recursivas o software projetado encontrará a resolução para o Sudoku, testando suas posições bem como analisando a possibilidade de o presente número pertencer de fato a posição inserida.

Como o tópico escolhido aborda os conceitos vistos na disciplina

O conhecimento de linguagens de programação é de fundamental importância para os programadores – bem como aos discentes – entretanto, apenas isto não capacita suficientemente e de maneira adequada. Tendo isto em vista, é imprescindível dispor de conhecimentos de modo a organizar estruturalmente os dados, ou seja, a estrutura de dados é essencial. Logo, podemos afirmar que para desenvolver e projetar o software que possui como pressuposto a resolução do Sudoku, a presente matéria é indispensável.

Outrossim, quando pensamos no conceito de Sudoku e de como sua resolução ocorre podemos tirar algumas conclusões como, por exemplo, a mesma análise é realizada por toda matriz com o intuito de completa-la corretamente. Tendo isto em vista, é possível averiguar um padrão de ocorrências, onde a mesma sempre realiza a mesma função, ou seja, quando

pensamos computacionalmente este conceito é facilmente relacionado à definição de uma pilha de chamadas recursivas – ou como também é conhecido Backtracking.

Ademais, a fim de elucidar a conceituação citada iremos dispor de uma breve descrição do mesmo. Concomitantemente, como os conceitos de Backtracking e recursividade são dependentes começaremos nossa explanação com uma ordem lógica, com a finalidade de esclarecer como ocorre o mesmo. A recursividade consiste em uma função ou método, o qual pode se invocar indefinidas vezes, sendo que a mesma pode ocorrer de maneira direta ou não. Aliás, com este conceito em mente ainda podemos começar a pensar no Backtracking, onde o mesmo consiste em uma pilha de chamadas recursivas, ou seja, uma série de chamadas recursivas serão feitas em sequência – ou não.

Todavia, considerando os conceitos apresentados e pensando computacionalmente no funcionamento do presente programa proposto, é notória a relação e a forma como a mesma ocorre. A cada nova posição que será preenchida na matriz do Sudoku, uma nova série de possibilidades serão apresentadas – entretanto, apenas uma será realmente válida – assim com o passar e desenrolar do jogo alterações precisaram ser realizadas no tabuleiro – nas posições que preenchemos – e com isso, teremos que voltar as posições anteriores e realizar novamente o processo de teste de valores, ou seja, dispomos de atos recursivos.

Portanto, o Backtracking traz um refinamento na busca realizada pelo software. Pois, sempre que necessitamos recorrer a uma função recursiva tanto os parâmetros quanto as suas variáveis locais utilizadas e passadas para a função no momento, serão empilhadas em uma pilha de execução – onde as variáveis locais são independentes entre si de modo a não perder os valores dos parâmetros das chamadas anteriores. Assim, se faz notório que com o auxílio de funções recursivas atingimos um refinamento de buscas bem como dispomos de um código mais estruturado, pois há poucos métodos distintos e o código está relacionado entre si.

Descrição dos testes

Quando pensamos computacionalmente é fácil perceber que o Sudoku é uma matriz – onde seguindo o seu tamanho padrão – dispõe de nove linhas e colunas. Sendo assim, a principal estratégia encontrada para resolução do problema apresentado foi o seu fracionamento em partes menores, com objetivo de ter um maior controle do funcionamento do programa bem como sua garantia de qualidade. Portanto, o primeiro ponto desenvolvido foi a constituição de uma matriz de tamanho apropriado e composta por zeros.

Ademais, visando garantir o cumprimento de sua principal característica – a de não haver valores repetidos – é realizado uma série de verificações, onde as mesmas se concentram na função *naoHaViolacao*. Tomando como base a função mencionada, é factível que a mesma dispõe de métodos apropriados os quais visam fazer uma análise por completo em suas linhas, colunas e submatriz, sendo assim caso o retorno desta função – a qual é uma booleana – seja *true* os valores estão de acordo, caso contrário precisamos utilizar outra função de modo a retomar à posição anterior, sendo que a mesma será explicada adiante.

Não obstante, é viável que neste ponto discorramos sobre o como os valores são selecionados computacionalmente, com o intuito de inseri-los na matriz. O tabuleiro que dispomos inicialmente é preenchido manualmente, ao ser selecionar no menu inicial, sendo

que concomitantemente a este fato estas posições preenchidas recebem o valor booleano de *true* – o qual auxilia na identificação das posições, pois as demais estarão como *false* - tendo isto em mente, foi desenvolvida a função denominada *computaSudoku*. Onde, o seu funcionamento é baseado na tentativa da inserção de valores, sendo assim designamos uma variável inteira intitulada *tentativa* – começando em um e que pode atingir até o valor nove, quando necessário – sendo que a cada nova posição vazia, a mesma tenta inserir um valor, caso seja válido passamos para a próxima posição até o completo preenchimento da matriz. Entretanto, caso o mesmo não ocorra, utilizaremos uma função – que como mencionado acima – retoma a posição anterior.

Outrossim, quando relacionamos o conceito de teste de valores bem como o retorno a posições anteriores a fim de encontrar a melhor opção, dispomos de uma busca refinada. Assim, quando precisamos trocar valores devido à violação das regras indiretamente estamos realizando está refinação, a qual possui como principal objetivo encontrar o melhor valor para determinada posição, ou seja, o Backtracking se faz presente. Nesta parte do código, nomeamos a mesma como *realizaBackTracking* onde a mesma fará com que nossa posição na matriz retroceda, caso seja necessário, até satisfazer o proposto, aliás a mesma pode ser invocada diversas vezes condicionando na pilha de chamadas recursivas juntamente com a *computaSudoku* – a qual, é recursiva.

Portanto, é notório o processo computacional que ocorre ao realizar um Sudoku bem como suas especificações. Logo, é necessário mencionar igualmente que a ideia inicial selecionada pelos discentes – números e posições aleatórias – não correspondeu aos resultados esperados. Sendo que, além do alto custo computacional a mesma interrompia nosso programa após o preenchimento da primeira linha do jogo apenas. Tendo ciência do exposto tomamos a liberdade de realizar algumas modificações com o intuito de continuar utilizando o conceito selecionado.

Solução encontrada

O custo computacional, consiste na eficiência e a quantidade de recursos que o algoritmo utiliza com o objetivo de realizar determina tarefa solicitada. Com base nisto, podemos averiguar que o número de operações que o algoritmo realizará faz uma estimativa de seu custo – tendo consciência que o mesmo depende igualmente da máquina bem como o sistema operacional utilizado. Assim, se faz notório que para o software proposto a utilização de números aleatórios não foi uma decisão correta, pois há diversas formas distintas de executar a mesma função através do código apresentado – mais baratas computacionalmente e que se aplicam ao exposto.

Logo, com o intuito de ter a plena convicção de que o código desenvolvido estava planejado e projetado corretamente, foram realizados alguns testes. Como comentado anteriormente, nossa execução era interrompida após o preenchimento da primeira linha do Sudoku, assim partimos do pressuposto de revisar e analisar o funcionamento – com base em testes – das seguintes funções: *computaSudoku* e *realizaBackTracking*, entretanto, nosso problema não se aplicava a estes conceitos. Ademais, através de pesquisas chegamos à conclusão que poderia haver um vazamento de memória devido à alta taxa de testes que o mesmo realizava para determinados jogos propostos aleatoriamente, bem como inválidos.

Assim, com o propósito de testar nosso programa – com base em valores os quais, sabíamos de antemão ter soluções concretas – alteramos a forma como a mesma era preenchida, com o intuito de ter consistência em nossa análise sobre o mesmo. Concomitantemente, como já previsto pelos discentes, o código funcionava corretamente e com um bom tempo de execução – bem como uma resolução satisfatória – para jogos tangíveis. Logo, partimos do pressuposto de mudar nossa maneira de compor o tabuleiro, o qual seria analisado computacionalmente bem como acrescentando uma parte de interação com o usuário.

Portanto, o artificio desenvolvido foi a entrada de valores via teclado pelo usuário. Ou seja, partimos do pressuposto que o usuário irá colocar um tabuleiro válido bem como seu objetivo será conferir se seus valores estão corretos além de dispor da melhor resolução – tendo em mente que o mesmo realiza uma busca refinada – ou seja, assim temos a certeza de a matriz existe bem como é resolvível e não gerará um custo exacerbado. Ademais, é importante salientar que apesar de ter sido desenvolvido e projetado com o propósito de dispor da melhor resolução assim como adequada, caso seja acrescido algum valor indevido bem como valores não condizentes com a proposta do software há a possibilidade de que o mesmo não consiga computar os valores. Logo, para o bom funcionamento do mesmo bem como sua eficiência é necessário seguir a lógica proposta.

Referência utilizadas no desenvolvimento

[A] COUTINHO, Demétrios. Site do Instituto Federal, Curitiba – PR, Brasil,

Acessado em 24/04/2021, às 10:34h.

http://www2.ouropreto.ifmg.edu.br/tp/slides/01-recursividade

[B] PARDO, Thiago. ICMC USP, Curitiba – Pr, Brasil,

Acessado em 28/04/2021, às 09:14h.

http://wiki.icmc.usp.br/images/f/f7/SCC-501-Aula_6_-_An%C3%A1lise_de_algoritmos_-_parte_1.pdf

[C] CELES, CERQUEIRA E RANGEL. Introdução a Estrutura de Dados. Editora Campus, 11º edição.