Trabalho 1: Solver de Sudoku via Força Bruta

Nomes: Henrique Fugie de Macedo e Luiz Otávio Silva Cunha
Matrículas: 0056151 e 0056153

explicará brevemente o problema abordado(feito); explicará em detalhe sua modelagem do problema para o paradigma utilizado; indicará os pontos fortes e fracos do algoritmo(feito).

SUDOKU

Sudoku é um jogo bastante conhecido mundialmente, criado em 1979 pelo arquiteto Howard Garns dos Estados Unidos, o jogo tem uma premissa e regras bem simples, necessitando apenas de um papel e caneta para ser jogado, o jogo contém um quadrado grande e dentro deste têm 9 quadrados menores, os quadrantes, cada quadrante tem área de 9 quadradinhos que se encontram dentro de cada quadrante.

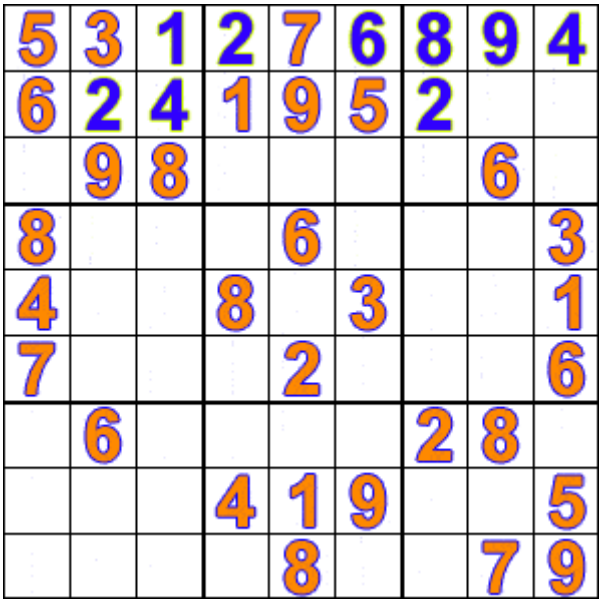
O objetivo do jogo é bem simples, apenas preencher todos os quadradinhos utilizando números de 1 a 9, porém a certas regras:

1. Dentro de cada quadrante, nenhum número deve-se repetir, ou seja se existe um número 7 dentro de um quadrante o 7 não poderá ser utilizado para preencher outro quadrado que se encontra no mesmo quadrante.

2. Em cada linha, nenhum número deve-se repetir, ou seja se existe um número 7 em uma linha o 7 não poderá ser utilizado para preencher outro quadrado que se encontra na mesma linha.

3. Em cada coluna, nenhum número deve-se repetir, ou seja se existe um número 7 em uma coluna o 7 não poderá ser utilizado para preencher outro quadrado que se encontra na mesma coluna.

4. O jogador pode apagar e reescrever qualquer número que ele tenha colocado, o jogo não tem vidas e só acaba quando o jogador desistir ou completar o tabuleiro seguindo as regras acima, porém ao jogador só é permitido apagar números que ele escreveu, sendo assim vetado poder apagar os números que já vem com o tabuleiro.



Como se pode reparar na imagem acima de um tabuleiro de sudoku, ele já vem preenchido com número não apagáveis e já contêm as divisórias para analisar a linha, coluna e o quadrante

Estratégia do SUDOKU

A várias estratégias para um jogo de sudoku, uma delas consiste em isolar o problema em três partes, colunas, linhas e quadrantes e assim que colocar um número verificar se esse número não se repetiu na linha, na coluna e no quadrante.

Backtracking

O Backtracking é um excelente algoritmo para resolver um jogo de sudoku, pois não é tão demorado igual a força bruta padrão que testa todas as possibilidades, o Backtracking recebeu esse nome pois o algoritmo fica refazendo seus passos para testar certas possibilidades, porém ao contrário da força bruta, o backtracking, não testa todas as possibilidades, pois se uma possibilidade já tem algum erro nela. não destas as seguintes daquela possibilidade.

Exemplificando melhor o backtracking no Sudoku, por exemplo suponha que o algoritmo resolve testar o número 1 em alguma posição não preenchida, coloca-se então o 1 nessa posição e verifica se está tudo certo, caso esteja errado o backtracking não insiste no erro e continua deixando o 1 alí, pois se já está errado não tem como no futuro magicamente dar certo, então o algoritmo descarta essa possibilidade, o que difere da força bruta padrão, que continuaria a insistir mesmo já estando errado.

O Backtracking é bom pois ele sempre chega no resultado, garantindo assim uma resposta, claro que a resposta deve existir, pois se não ficará testando infinitamente, mas provada a existência da resposta o backtracking se implementado de forma correta garante encontra-lá.

Por fim, o backtracking não apresenta uma estratégia de execução, exemplificando melhor, no sudoku suponha que a melhor estratégia é começar pelo meio, se o programador não programar assim o backtracking começa do início e vai executando, logo ele não apresenta a necessidade de uma preempção.

Concluí-se que com tudo que foi apresentado, o backtracking apresenta vários pontos positivos e negativo, como qualquer algoritmo de computação, porém para problemas semelhantes ao sudoku, que exigem o teste de muitas possibilidades o backtracking é um algoritmo ideal.

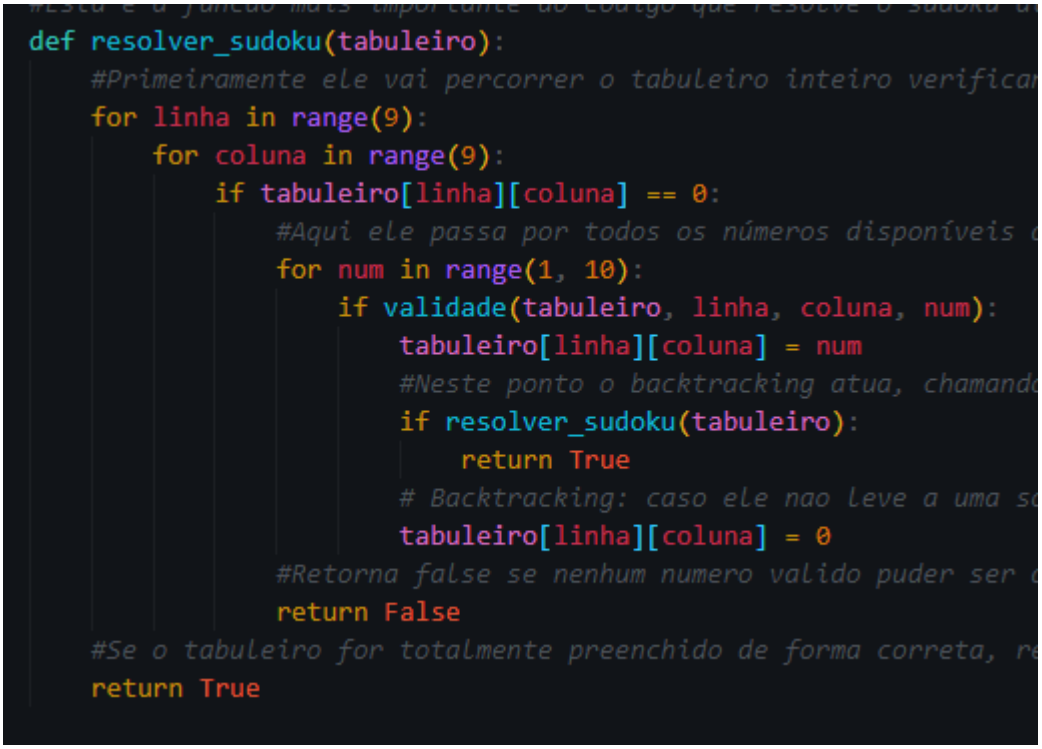
Explicação do código e modelagem utilizada:

O código feito para solucionar o sudoku começa mostrando um menu que permite o usuário escolher em um Menu, se ele quer escolher manualmente um dos sudoku`s presentes na pasta ou escolher aleatoriamente um deles, onde o arquivo está dentro do padrão fornecido no trabalho e ele é armazenado em uma lista de listas, onde os valores são armazenados em int, pois mais a frente em algumas verificações, como a verificação dos quadrantes. Antes da leitura, também é feita uma verificação, com a função tabuleiro_valido(), solicitada no enunciado do trabalho com a finalidade de garantir que o tabuleiro fornecido não possui nenhum erro em relação às regras do sudoku.

Em seguida, após receber o sudoku, ele mostra o tabuleiro lido para verificação, para mostrar que o tabuleiro lido é o escolhido e começa um timer, para calcular o tempo que o programa demorou para solucionar o sudoku.

Alguns pontos importantes, o tabuleiro do sudoku está sendo resolvido da primeira posição do tabuleiro 1x1 até a 9x9, percorrendo a primeira linha inteira, para em seguida ir para a segunda linha até chegar ao final e em relação aos quadrantes, a forma para percorrer cada quadrante corretamente, ele pega a linha e a coluna da posição e em seguida divide por 3, recebendo o número inteiro dessa divisão, para encontrar em qual quadrante ele está, por exemplo, a linha 4 e coluna 4, tem resto 1, tanto a linha quanto a coluna, e multiplica por 3, para ajustar a posição no tabuleiro, então ele vai pegar o quadrante da segunda linha e da segunda coluna, na posição à esquerda em cima, para percorrer até o final do quadrante e verificar ele inteiro tanto na verificação da validade do tabuleiro quando na hora de preencher alguma célula.

Explicando agora como o backtracking do código funciona, como é feito, vamos começar mostrando como é o código utilizado:

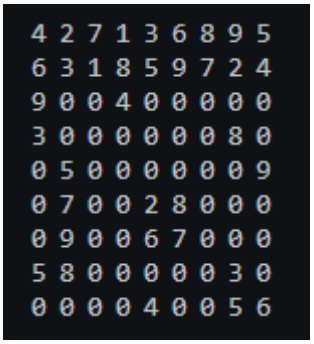


```
def resolver_sudoku(tabuleiro):
    #Primeiramente ele vai percorrer o tabuleiro inteiro verifica
    for linha in range(9):
        for coluna in range(9):
            if tabuleiro[linha][coluna] == 0:
                #Aqui ele passa por todos os números disponíveis
                for num in range(1, 10):
                    if validade(tabuleiro, linha, coluna, num):
                        tabuleiro[linha][coluna] = num
                        #Neste ponto o backtracking atua, chamand
                        if resolver_sudoku(tabuleiro):
                            return True
                        # Backtracking: caso ele nao leve a uma s
                        tabuleiro[linha][coluna] = 0
                #Retorna false se nenhum numero valido puder ser
                return False
    #Se o tabuleiro for totalmente preenchido de forma correta, r
    return True
```

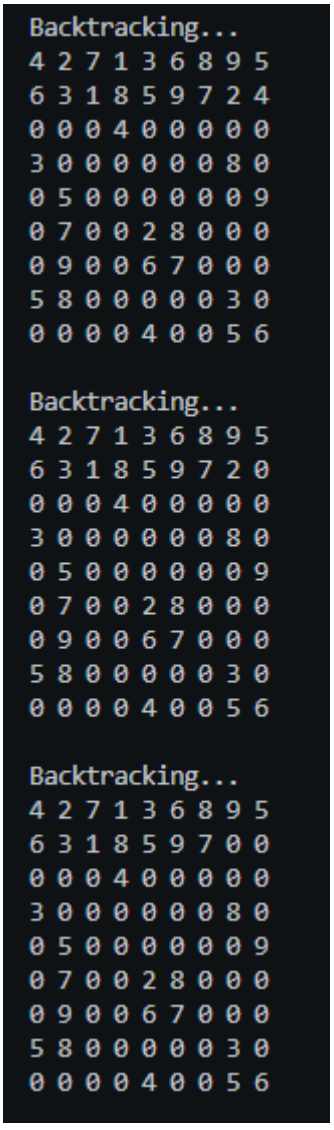
Na função, ela começa com dois for, para percorrer o tabuleiro de sudoku, seguido de uma verificação do valor na célula do tabuleiro, para verificar se é uma posição vazia(com valor 0 na célula), se sim, ele tem outro for para percorrer todos os números possíveis em um sudoku, percorrendo entre os números 1 a 9. Neste ponto, para impedir que ele tente preencher um número que não seria válido, ele faz uma verificação com a função validade, que verifica se na linha/coluna/quadrante não possui o número que vai ser preenchido. Se o número passar nesta validade ele é armazenado na posição do tabuleiro e em seguida é feita uma verificação com uma chamada recursiva, é nesta parte que o backtracking começa, pois nesta verificação, caso ela retorne True, quer dizer que a tentativa de preencher a célula foi bem sucedida, caso contrário, ele vai tentar um próximo número no for de 1 a 9, tentando todas as possibilidades de preenchimento do tabuleiro e também é responsável por encontrar um ramo que possui a solução do sudoku. Caso o sudoku encontre um beco sem saída, que seria basicamente quando a recursão, na procura de uma solução do sudoku, chega em um caso em que não tenha nenhum número válido para preencher, o backtracking atua retornando para uma célula anterior e atribuindo valor 0 na célula travada, para que ele explore outros ramos de solução até encontrar um tabuleiro válido e completamente preenchido.

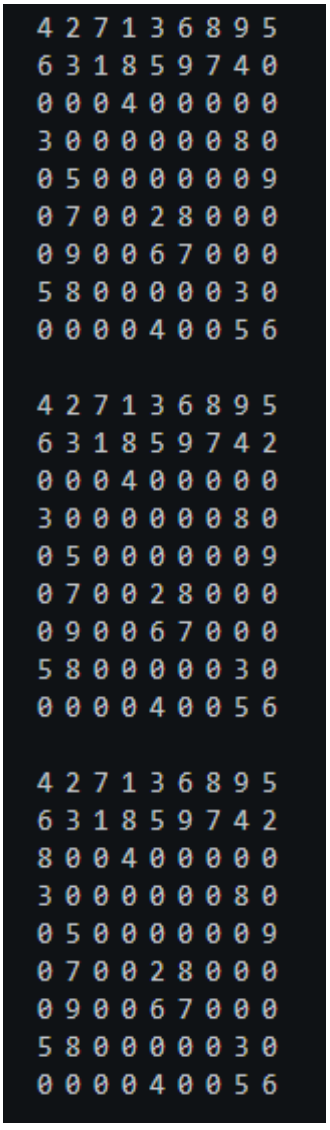
Explicando de curto modo, ele percorre o tabuleiro completo, verificando as posições vazias, passando por todos os números possíveis que podem ser colocados dentro da célula atual e através de uma recursão, ele vai avançando para as demais células para tentar preencher, e caso ele chegue em um ponto que não seja possível continuar, um beco sem saída que não tenha nenhum número válido, ele retorna a célula para o estado inicial, valor 0, e tenta um caminho diferente. Aqui está uma imagem mostrando ele atingindo um

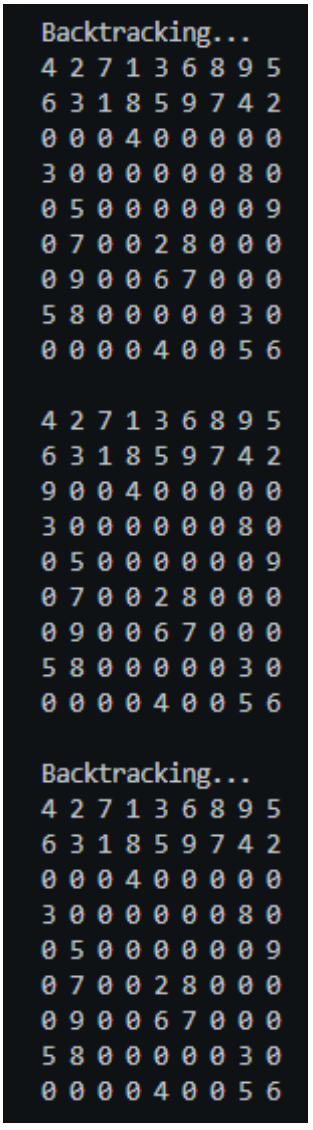
ponto onde não possui mais possibilidades e ele executa o backtracking para tentar um caminho diferente:



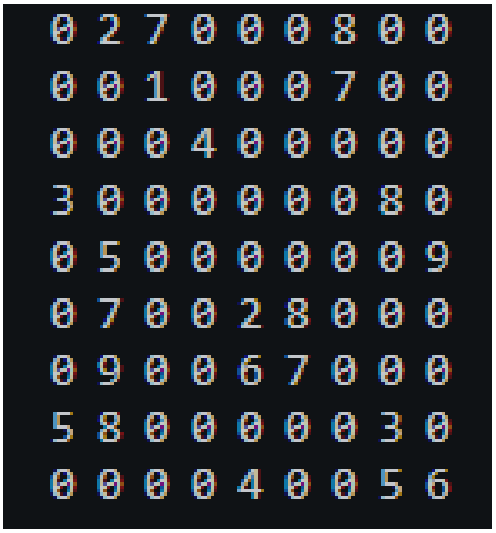
Este é um momento de execução do código, onde ele preencheu um 9, o último número do for, e não pode preencher o 1 pois o quadrante já possui 1, sendo obrigado a fazer backtracking para tentar achar outro caminho.

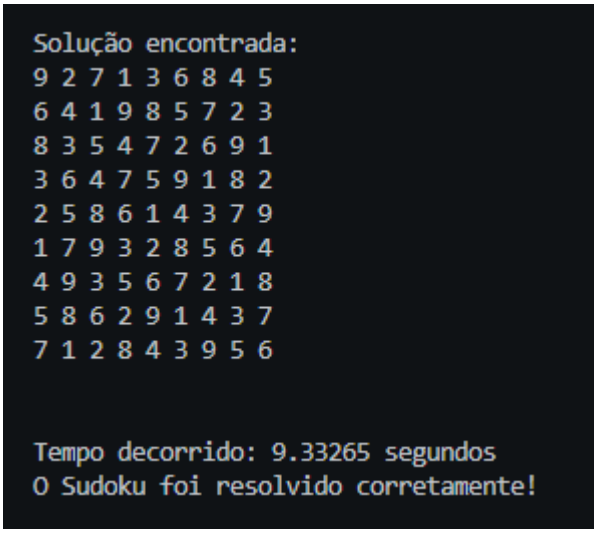






A imagem da esquerda mostram os 3 passos seguintes da imagem anterior, onde foi feito o backtracking e foi removido o 9, e como não existe nenhuma outra possibilidade de número para ser colocado, foi executado um segundo backtracking, removendo o número 4 no final da segunda linha e novamente, não é possível colocar nenhum outro número além do 4, sendo necessário um terceiro backtracking para remover o 2, e neste caso existe um número possível além do 2. Agora a imagem do meio mostra os 3 passos seguintes, preenchendo o outro número além do 2, o 4, que poderia ser preenchido, e a execução continua até chegar em outro beco sem saída, após preencher o 8 na primeira coluna da terceira linha. A imagem da direita mostra novamente outro caso de backtracking e mostra como ele é um algoritmo de forma bruta, pois ele remove o 8 e em seguida preenche com 9, fazendo outro backtracking removendo ele novamente, ele testa todas as possibilidades de caminho até encontrar um caminho válido, um algoritmo mais eficiente não testaria este caso com o 9.





Para finalizar, esta é a solução do sudoku fornecido, e a esquerda o tabuleiro limpo, neste caso, o tempo foi bem elevado, e foi devido o tanto de ramos percorrido, pois na demonstração de backtracking, a primeira casa foi preenchida com o valor 4, e a solução final, a primeira casa possui o valor 9, ou seja, ele removeu o 4, preencheu com 5, removeu, até chegar no número 9 e foi seguindo até chegar no resultado final e resolver o sudoku.