Universidade Federal do Rio de Janeiro
Bacharelado em Ciência da Computação
Lógica

Documentação referente ao algoritmo solucionador de Sudoku

Eduardo da Silva Barbosa - 116150432
Rafael Pais Cardoso - 116140788
Tainá da Silva Lima - 116165607
2018

Introdução

Sudoku é um quebra-cabeça lógico, sendo nada mais que um caso especial de quadrados latinos[1]. O jogo tem como objetivo preencher todas as casas de uma matriz NxN com números de 1 a N. Existem inúmeras variações de como o tabuleiro é construído, porém o mais conhecido é aquele em que a matriz possui 9 linhas e 9 colunas, e as seguintes condições devem ser respeitadas:

- Para cada casa existe pelo menos um número na mesma
- Para cada casa existe no máximo um número na mesma
- Para cada linha, não pode aparecer um número mais de uma vez
- Para cada coluna, não pode aparecer um número mais de uma vez
- Para cada sub matriz 3x3, não pode aparecer um número mais de vez

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figura 1: Um exemplo de sudoku

[1] Um quadrado latino de ordem n é um arranjo de N símbolos diferentes. Tais símbolos são organizados em uma matriz NxN, onde cada símbolo só aparece uma única vez em cada linha e uma única vez em cada coluna. Tal conceito é muito utilizado na área de estatística e álgebra, com aplicações em planejamento de experimentos e generalização de grupos.

Objetivo

O algoritmo tem como objetivo achar uma solução para o sudoku, que é dado como entrada, utilizando como base os conceitos de lógica proposicional apresentados em sala de aula.

Modo de uso

O arquivo de entrada

O usuário dará como entrada um arquivo na extensão .txt (Forma.txt) que conterá os fatos do sudoku, ou seja, quais números estão em quais posições inicialmente. A entrada consiste de 3 dígitos: o primeiro a posição que ele ocupará em uma determinada linha, o segundo a posição que ocupará em uma determinada coluna e o terceiro representa o número a ser colocado na casa. Um exemplo de entrada seria o número 237 (2 representa a posição na linha, 3 a posição na coluna e 7 o valor a ser colocado na casa).

Após inserir a primeira sequência de 3 dígitos, pula-se uma linha e o outro é escrito abaixo deste e assim por diante.

A execução

O usuário irá abrir o terminal e navegar até a pasta em que se encontra o arquivo “Trabalho_Sudoku.zip”. Irá descompactá-lo e em seguida, irá compilá-lo com o seguinte comando escrito no terminal: “gcc -o main main.c -O3 -Wall -ansi”.

Para executá-lo, basta digitar no terminal o comando “./main”.

Funcionamento

A lógica

Como dito anteriormente, sudoku é basicamente um problema lógico, sendo assim, suas restrições podem ser mapeadas para fórmulas bem formadas (fbf), utilizando um conjunto de proposições "Prop" apresentados em aula:

$$Prop = \{x_{ijd} | 1 \leqslant i \leqslant 9, 1 \leqslant j \leqslant 9, 1 \leqslant d \leqslant 9\}$$

Figura 2: Conjunto das proposições

onde x_{ijd} representa "Ter o número d na posição ij".

Tais fórmulas[2] são:

- Restrição onde cada casa tem que possuir pelo menos um número:

$$A = \bigwedge_{i,j \in D} A_{ij}$$

$$A_{ij} = (x_{ij1} \vee x_{ij2} \vee x_{ij3} \vee x_{ij4} \vee x_{ij5} \vee x_{ij6} \vee x_{ij7} \vee x_{ij8} \vee x_{ij9})$$

Figura 3: fbf para a primeira restrição

- Restrição onde cada casa tem que possuir no máximo um número:

$$B = \bigwedge_{d,k \in D} B_{ij}$$

$$B_{ij} = \bigwedge_{d,k \in D, d \neq k} \neg(x_{ijd} \wedge x_{ijk})$$

Figura 4: fbf para a segunda restrição

[2] Fórmulas retiradas do slide 6 disponibilizado pelo Prof. João Carlos menos a última (referente ao subgrid).

- Restrição onde cada número não aparece mais de uma vez na mesma linha:

$$C = \bigwedge_{i,d \in D} C_{id}$$

$$C_{id} = \bigwedge_{j \in D} \bigwedge_{k \in [j+1,9]} \neg(x_{ijd} \wedge x_{ikd})$$

Figura 5: fbf para a primeira restrição

- Restrição onde cada número não aparece mais de uma vez na mesma coluna:

$$E = \bigwedge_{j,d \in D} E_{jd}$$

$$E_{jd} = \bigwedge_{i \in D} \bigwedge_{k \in [i+1,9]} \neg(x_{ijd} \wedge x_{kjd})$$

Figura 6: fbf para a segunda restrição

- Restrição onde cada número não aparece mais de uma vez na mesma sub-matriz 3x3:

$$S_1 = \wedge_{d \in D; k,t \in \{1,2,3\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i,j \in \{1,2,3\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$S_2 = \wedge_{d \in D; k \in \{1,2,3\}, t \in \{4,5,6\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i \in \{1,2,3\}, j \in \{4,5,6\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$S_3 = \wedge_{d \in D; k \in \{1,2,3\}, t \in \{7,8,9\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i \in \{1,2,3\}, j \in \{7,8,9\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$S_4 = \wedge_{d \in D; k \in \{4,5,6\}, t \in \{1,2,3\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i \in \{4,5,6\}, j \in \{1,2,3\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$S_5 = \wedge_{d \in D; k \in \{4,5,6\}, t \in \{4,5,6\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i \in \{4,5,6\}, j \in \{4,5,6\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$S_6 = \wedge_{d \in D; k \in \{4,5,6\}, t \in \{7,8,9\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i \in \{4,5,6\}, j \in \{7,8,9\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$S_7 = \wedge_{d \in D; k \in \{7,8,9\}, t \in \{1,2,3\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i \in \{7,8,9\}, j \in \{1,2,3\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$S_8 = \wedge_{d \in D; k \in \{7,8,9\}, t \in \{4,5,6\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i \in \{7,8,9\}, j \in \{4,5,6\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$S_9 = \wedge_{d \in D; k \in \{7,8,9\}, t \in \{7,8,9\}} A_{k,t,d}$$
$$A_{k,t,d} = \wedge_{i \in \{7,8,9\}, j \in \{7,8,9\}} \neg (x_{ktd} \wedge x_{ijd})$$

$$F = \wedge_{m \in \{1,\ldots,9\}} S_m$$

onde para cada par (k,t) mencionado nas fórmulas S, (k,t) é diferente de (i,j).

Figura 7: fbf para a terceira restrição

Representando as proposições e o tabuleiro solução

Os valores verdades (0 - falso e 1 - verdadeiro) das proposições são armazenadas na matriz de 3 dimensões “matrix_prop[i][j][k]” que está organizada da seguinte maneira: as duas primeiras dimensões representam a posição (i,j) no tabuleiro e a terceira (k) representa o número que está naquela casa, ou seja, se em matrix_prop[1][6][3] = 1, isso nos diz que na posição (1,6) do tabuleiro há o número 3. Ambas as dimensões referentes à posição no tabuleiro variam de 0 à 8 e a terceira computacionalmente varia de 0 à 8, ou seja, seria possível colocar 9 viáveis valores em cada um dos índices da matriz, no entanto, em código, foi feita uma modificação para que variasse de 1 à 9 como é na teoria.

A variável “matrix[i][j]” é uma matriz extra que é utilizada apenas para fins de facilidade de impressão da solução, ela guarda os números de fato (1 à 9) em cada posição do tabuleiro.

Uma variável do tipo Solution guarda a matriz de proposições que soluciona o Sudoku em questão, sendo organizada da mesma maneira que a matriz “matrix_prop[i][j][k]”.

Gerando as fórmulas bem formadas que regem o Sudoku

Cada fbf representa uma regra do Sudoku, como mencionado previamente, e para que um tabuleiro seja uma solução do Sudoku, ele deve satisfazer tais fbfs.

Portanto, ao migrarmos para a implementação computacional é natural pensarmos em criar métodos que fazem o papel dessas regras. E assim foi feito:

- A regra “em cada casa existe pelo menos (no mínimo) um número” é representada pelo método “check_MinDig_positions”
- A regra “em cada casa existe no máximo um número” é representada pelo método “check_MaxDig_positions”
- A regra “para cada linha, não pode aparecer um número mais de uma vez” é representada pelo método “check_Rows”
- A regra “para cada coluna, não pode aparecer um número mais de uma vez” é representada pelo método “check_Cols”

- A regra “para cada subgrid 3x3, não pode aparecer um número mais de uma vez” é representada pelo método “check_Subgrid”

Cada um desses métodos recebem como parâmetro a matriz das proposições (matrix_prop[N][N][N]) a qual será consistente ou não.

Como essas fbfs envolvem muitas proposições, a codificação das mesmas para métodos se torna inviável de ser feita manualmente, por isso foram criados algoritmos para criar essas fbfs. Esses algoritmos se localizam nos arquivos “formula1.c”, “formula2.c”, “formula3.c”, “formula4.c” e “formula5.c”, respectivamente. De maneira resumida, cada um desses algoritmos geram uma string, utilizando comandos “for” aninhados para percorrer os índices da matriz de proposições, onde essa string é quem contém a fbf, que no final é escrita nos arquivos .txt com mesmo nome do arquivo em .c.

Por fim, os conteúdos dos arquivos .txt são copiados para as funções criadas no arquivo “main.c” correspondentes às suas regras.

Encontrando a solução do Sudoku

O processo da execução do código ocorre da seguinte forma:

- As matrizes matrix[N][N] e matrix_prop[N][N][N] são inicializadas com 0, através das funções init_matrix2D e init_matrix3D, respectivamente.
- As entradas serão lidas do arquivo .txt (Forma.txt) a partir da função read_numbers. Onde a mesma chama a função save_number que será responsável por preencher a variável matrix com os valores nas posições correspondente e a matrix_prop com 1’s nas proposições correspondente.
- O conteúdo da variável matrix será impressa, através da função print_grid. Nesse momento a matriz preenchida será impressa no console com os valores de entrada nas posições correspondentes obtidos no arquivo .txt.
- A função find_solution será chamada para resolver o sudoku inserindo números de forma aleatória nas posições vazias da matriz. Desse modo, cria-se uma matriz temporária, que

é uma cópia da matriz matrix_prop, em que números de 0 a 8 são obtidos de forma aleatória. As posições (ij) vazias da matriz matrix e o valor obtido randomicamente (d) fazem referência às proposições ijd da Figura 2. Assim, a matriz temporária será preenchida com 1s nas posições ijd (i - linha, j - coluna, d - dimensão) obtidas. Após preencher todas as posições vazia,, são chamadas as funções check_MinDig_positions, check_MaxDig_positions, check_Rows, check_Cols e check_Subgrid. Essas funções representam as fórmulas bem formadas das Figuras 3 até a 7 e só chegaremos na solução do sudoku caso todas essas fórmulas retornem verdadeiro. Caso a solução não seja encontrada, a matriz temporária recebe novamente a cópia da matrix_prop e novos número aleatórios são gerados para preenchê-la. Isso é feito até conseguir encontrar alguma possível solução.

Obs. 1: Todas as funções citadas neste tópico estão melhor descritas no Anexo I.

Obs. 2: Os testes para encontrar soluções estão listados no Anexo II.

Ambiente de desenvolvimento

O contexto de elaboração desse algoritmo foi:

- Sistema Operacional: Ubuntu versão 18.4.1 LTS
- Editor de texto: Atom
- Linguagem de programação: C
- Compilador: gcc (Ubuntu 7.3.0-16ubunto3) 7.3.0

Referências bibliográficas

- CONCEPTIS PUZZLE. **Sudoku History**. Disponível em: <https://www.conceptispuzzles.com/index.aspx?uri=puzzle/sudoku/history>. Acesso em: 21 de setembro de 2018
- WIKIPEDIA. **Latin square**. Disponível em: <https://en.wikipedia.org/wiki/Latin_square>. Acesso em: 21 de setembro de 2018.
- PEREIRA, B. et al. **Quadrados Latinos**. Universidade Estadual de Campinas, Campinas, 2011. Disponível em: <https://www.ime.unicamp.br/~ftorres/ENSINO/MONOGRAFIAS/EA_2011_MONOI.pdf>. Acesso em: 21 de setembro de 2018
- WIKIPEDIA. **Sudoku**. Disponível em: <https://en.wikipedia.org/wiki/Sudoku>. Acesso em: 21 de setembro de 2018
- FRONTULL, Samuel. **Applications of Propositional Logic**. 2017. 20 p. Relatório de Seminário. Disponível em: <http://cl-informatik.uibk.ac.at/teaching/ws16/plar/reports/SF.pdf>. Acesso em: 21 setembro 2018

Anexo I - Descrição das funções

Função “print_grid”

Parâmetros: int grid[N][N] (Matriz 9x9 do tabuleiro).

Objetivo: Imprimir o tabuleiro no console.

Retorno: Não retorna.

Função “save_number”

Parâmetros: int grid[N][N] (Matriz 9x9 do tabuleiro), int matrix_prop[N][N][N] (matriz 9x9x9 com as proposições), int number (número a ser armazenado na matriz 9x9 do tabuleiro).

Objetivo: Salvar o número passado como argumento na matriz do tabuleiro.

Retorno: Não retorna.

Função “read_numbers”

Parâmetros: int grid[N][N] (Matriz 9x9 do tabuleiro), int matrix_prop[N][N][N] (matriz 9x9x9 com as proposições).

Objetivo: Ler cada linha do arquivo e passar o números para a função save_number.

Retorno: Não retorna.

Função “init_matrix2D”

Parâmetros: int grid[N][N] (Matriz 9x9 do tabuleiro).

Objetivo: Inicializar a matriz do tabuleiro colocando o valor 0 em todas as posições.

Retorno: Não retorna.

Função “init_matrix3D”

Parâmetros: int grid[N][N][N] (Matriz 9x9x9 com as proposições).

Objetivo: Inicializar a matriz que representa as proposições com 0.

Retorno: Não retorna.

Função “check_Rows”

Parâmetros: int matrix_prop[N][N][N] (Matriz 9x9x9 com as proposições)

Objetivo: Checar a consistência da matriz, verificando se para cada linha da matriz de proposições não há números repetidos.

Retorno: Retorna verdadeiro(1) caso a matriz esteja consistente e retorna falso(0) caso a matriz não esteja consistente.

Função "check_Cols"

Parâmetros: int matrix_prop[N][N][N] (Matriz 9x9x9 com as proposições)

Objetivo: Checar a consistência da matriz, verificando se para cada coluna da matriz de proposições não há números repetidos.

Retorno: Retorna verdadeiro(1) caso a matriz esteja consistente e retorna falso(0) caso a matriz não esteja consistente.

Função " check_Subgrid"

Parâmetros: int matrix_prop[N][N][N] (Matriz 9x9x9 com as proposições)

Objetivo: Checar a consistência da matriz, verificando se para cada subgrid 3x3 da matriz de proposições não há números repetidos.

Retorno: Retorna verdadeiro(1) caso a matriz esteja consistente e retorna falso(0) caso a matriz não esteja consistente.

Função " check_MaxDig_positions"

Parâmetros: int matrix_prop[N][N][N] (Matriz 9x9x9 com as proposições)

Objetivo: Checar a consistência da matriz, verificando se para cada casa da matriz de proposições existe no máximo um dígito.

Retorno: Retorna verdadeiro(1) caso a matriz esteja consistente e retorna falso(0) caso a matriz não esteja consistente.

Função " check_MinDig_positions"

Parâmetros: int matrix_prop[N][N][N] (Matriz 9x9x9 com as proposições)

Objetivo: Checar a consistência da matriz, verificando se para cada casa da matriz de proposições existe no mínimo um dígito.

Retorno: Retorna verdadeiro(1) caso a matriz esteja consistente e retorna falso(0) caso a matriz não esteja consistente.

Função "find_solution"

Parâmetros: int matrix[N][N] (Matriz 9x9 do tabuleiro), int matrix_prop[N][N][N] (matriz 9x9x9 com as proposições).

Objetivo: Encontrar uma solução para o tabuleiro.

Retorno: Não há retorno.

Função " copy_matrix3D"

Parâmetros: int matrix_prop[N][N][N] (Matriz 9x9x9 com as proposições)

Objetivo: Fazer uma cópia da matriz enviada, guardando-a numa variável ponteiro do tipo Solution, retornando o mesmo.

Retorno: Retorna um ponteiro para um Solution.

Anexo II - Testes

Ambiente de testes

- Sistema Operacional: Ubuntu versão 18.4.1 LTS
- Processador: FX 8350
- RAM: 12 GB

Casos

- Caso 1: Faltando 1, 2, 3 ou 4 elementos (9, 81, 729, 6561 Possibilidades de solução, respectivamente)

Nº do teste	Tempo aproximado
1	Menor que 1 segundo
2	Menor que 1 segundo
3	Menor que 1 segundo
4	Menor que 1 segundo
5	Menor que 1 segundo

- Caso 2: Faltando 5 elementos (59.049 Possibilidades de solução)

Nº do teste	Tempo aproximado
1	Menor que 1 segundo
2	2 segundos
3	1 segundo

- Caso 3: Faltando 6 elementos (531.441 Possibilidades de solução)

Nº do teste	Tempo aproximado
1	5 segundos
2	20 segundos
3	3 segundos

- Caso 4: Faltando 7 elementos (4.782.969 Possibilidades de solução)

Nº do teste	Tempo aproximado
1	1 minuto e 12 segundos
2	20 segundos
3	38 segundos
4	3 segundos
5	10 segundos
6	24 segundos

- Caso 5: Faltando 8 elementos (43.046.721 Possibilidades de solução)

Nº do teste	Tempo aproximado
1	20 minutos
2	3 minutos
3	5 minutos

- Caso 6: Faltando 9 elementos (387.420.489 Possibilidades de solução)

Nº do teste	Tempo aproximado de espera	Obs.
1	41 minutos e 30 segundos	Teste cancelado devido a demora para encontrar uma solução