Trabalho Prático 3

Resolução de instâncias de Sudoku

Guilherme Mendes de Oliveira
Departamento de Ciência da Computação

Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

guilhermemendes@ufmg.br

1. Introdução

Este trabalho tem como objetivo avaliar uma instância de Sudoku e determinar se o mesmo tem ou não solução a partir da redução do problema de coloração de grafos. A partir de uma heurística, caso tenha solução deve fornece-la.

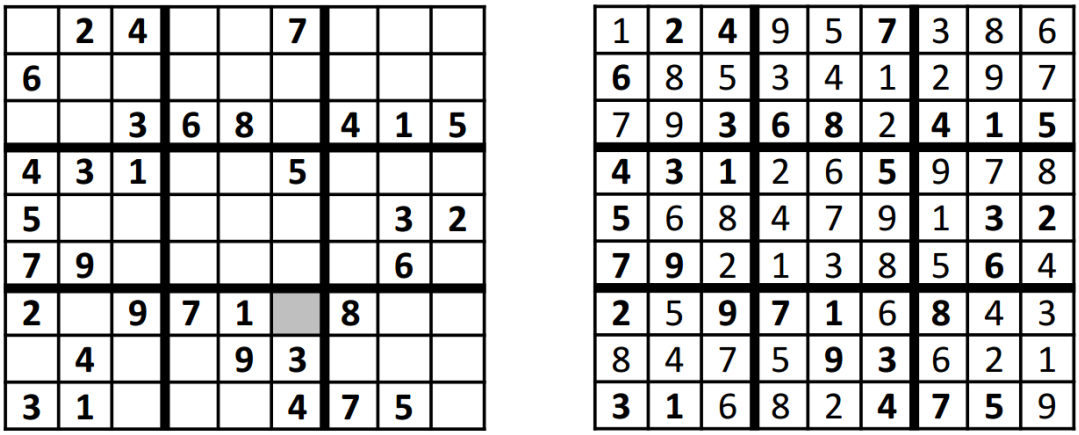

Figura 1 – Exemplo de Sudoku

Para tal, além do tratamento da instância deve se utilizar uma heurística relacionada a coloração de grafos para a tomada de decisão de qual vértice do grafo Sudoku deve ser "colorido" por vez

2. Implementação

2.1 Instruções de compilação e execução

O programa foi desenvolvido na linguagem C++ e compilado pelo G++ da Minimalist GNU for Windows (MinGW) no ambiente Linux distribuição Ubuntu versão 18.04.02.

2.2 Estrutura de Dados

Para a implementação foram utilizados dois tipos abstrato de dados.

Vértice: Representa o objeto a ser armazenado no Grafo, ele possui uma identificação e armazena sua posição no grafo, o valor da célula do Sudoku, as coordenadas de sua localização se o Sudoku for considerado como uma matriz, estes dados são importantes para a definição dos quadrantes, a saturação, que significa quantos vizinhos tem valores diferentes e uma lista de adjacência.

```
class Vertice{
    public:
    int idVertice;
    int valor;
    int cordI;
    int cordJ;
    int saturacao;
    bool color;
    vector<int> coresVizinhos;
    vector<Vertice> adj;
    Vertice(int idVertice, int valor);
    void printList();
    bool buscaValor(int valor);
};
```

Figura 2 – Implementação do TAD *Vértice*.

Graph: Representa o grafo Sudoku como um arranjo de Vértices, campos do Sudoku, e todos os métodos que o envolvam sua manipulação.

```
class Graph{
    public:
    int numVertices;
    int ladoSudoku;
    bool solucao;
    vector<Vertice> Vertices;

    Graph(int nVertices, int ladoSudoku);
    void addVertices(Vertice v);
    void addArestas(int idA, int idB);
    void verListaVertices();
    bool existeAresta(int idA, int idB);
    void coloreVertices();
    void calculaSat();
    bool verificaSolucao();


};
```

Figura 3 – Implementação do TAD *Graph*.

2.3 Modularização

Foram criados além dá *main* outro arquivo, graph.hpp que apresenta a estrutura dos TAD's utilizados para a implementação e o graph.cpp que corresponde a implementação dos métodos correspondentes aos TAD's.

2.4 Código

A função *main* é responsável por criar a o Grafo do Sudoku a partir de um arquivo de texto (*.txt) lido pela biblioteca *fstream* com as instruções para o tamanho do Sudoku, a quantidade de linhas e colunas dos quadrantes e a instancia.

Cria os vértices com suas respectivas listas de adjacências e atributos.

Além disso ela realiza a alocação dos Vértices do Sudoku no grafo para o tratamento através da Heurística de Saturação máxima para a ordem de escolha dos vértices para "colorir", ou seja aqueles vértices que possuem a menor quantidade de cores possível e não possuem cores, devem ser os primeiros a terem sua cor definida, a partir disso uma nova saturação é computada para os vértices até que todos os vértices estejam coloridos ou que não seja mais possível decidir pela atribuição de cores dentro das possibilidades dos que faltam.

3. Analise de Complexidade:

Por se tratar de um problema de NP-Completo, não é possível resolver um Sudoku vazio em tempo polinomial, no entanto a partir da instancia fornecida, podemos executar procedimentos a partir da heurística de saturação máxima em tempo polinomial, as funções que são responsáveis por instanciar o Grafo e os Vértices juntamente com seus atributos a computação da Saturação de cada vértice são O (n^2), a função responsável por calcular uma solução a partir da instancia tem custo O (n^3).

A medição do tempo de execução ficou prejudicada uma vez que o código não garante a corretude para o problema, portanto avaliar quais são as situações críticas para o problema proposto juntamente com a Heurística escolhida fica inviável.

4. Avaliação Experimental

Podemos concluir que:

O relacionamento entre os campos do Sudoku de fato é bem intuitivo ao tratarmos a organização do Sudoku como um grafo tendo os relacionamentos como a chave para a resolução via coloração de grafos, todos os vértices que estão na mesma linha, coluna e quadrante se relacionam.

A Heurística, no entanto, não é trivial e sua implementação não foi atingida pelo código, ao avaliarmos a solução necessária temos que levar em consideração de maneira decrescente a saturação dos vértices, e a cada coloração efetuar o recalculo o que não foi possível.

5. Conclusão

A implementação do Sudoku através de grafos é menos difícil do que se pensado antes de começarmos a modelar para partir para a implementação a parte mais complicada é a implementação da Heurística para tratamento das instâncias para a Saturação Máxima é preciso escolher os Vértices que tem as menores possibilidades de cores e a cada atribuição de cor recalcular a saturação e efetuar o mesmo procedimento, o que não foi possível, a solução apresentada não trata os problemas uma vez que a escolha via índice não garante a melhor sequência para atribuição de valores.

O resultado da implementação não foi bem satisfatório uma vez que foram utilizados muitos métodos e o código não apresenta solução correta para nenhum dos casos de teste. Ao observamos a **Análise de Complexidade** o custo operacional, isto é, desconsiderando o custo de implementação da estrutura, é polinomial tendo algumas funções com custo O (n), O (n^2) e uma O (n^3).

6. Referências

KLEINBERG, J; TARDOS, E. Algorithm Design. Capítulo 3: Graphs. Editora Pearson.

KLEINBERG, J; TARDOS, E. Algorithm Design. Capítulo 8: Intractability. Editora Pearson.

<https://www.ic.unicamp.br/~atilio/slidesWtisc.pdf/> Acesso em: 18 de Novembro de 2019