# UNIVERSIDADE FEDERAL DE MINAS GERAIS

**DCC005 - ALGORITMOS E ESTRUTURAS DE DADOS III**

**HERNANE BRAGA PEREIRA**

**2014112627**

**TRABALHO PRÁTICO - 3**

Belo Horizonte - MG
Julho/2018

# INTRODUÇÃO

O objetivo deste trabalho é a resolução do problema da empresa *BH Software*, criadora de um aplicativo de comunicação por áudio e vídeo que vem ganhando popularidade. Ela deseja resolver o problema de atualização em seus servidores, pois os mesmos possuem as seguintes restrições: não podem ser atualizados de um em um e os servidores adjacentes, diretamente conectados entre si, não podem ser atualizados ao mesmo tempo. A solução para este dilema é implementar um algoritmo que faça atualizações por rodadas e que respeite o critério de adjacência entre servidores.

# SOLUÇÃO DO PROBLEMA

Para solucionar o problema apresentado, o mesmo foi modelado como um grafo não direcionado que deve ser colorido de forma que vértices adjacentes não possuam a mesma cor. Foi pedido que o problema fosse resolvido utilizando duas abordagens distintas: força bruta e heurística.

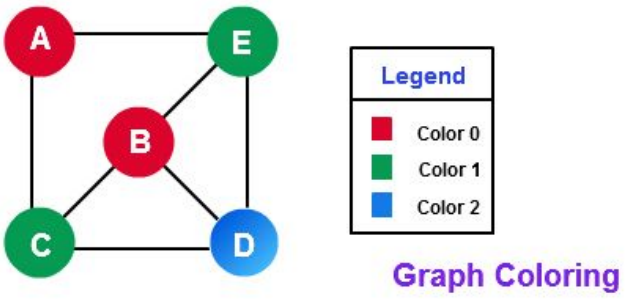


**Figura 1.** Exemplo de um problema de coloração em grafo

## 2.1 Solução do problema usando força bruta

Para solucionar o problema usando a abordagem de força bruta, foi preciso criar todas as possíveis soluções para o problema e testá-las uma a uma. Quando o algoritmo encontra uma solução que é possível ela é impressa nos arquivos *rodada.txt* e *alocacao.txt*, então o programa é finalizado.

```
Ler grafo de entrada;
Cria grafo usando lista de adjacências;
Cria vetor de cores[ ] de 0 até N = número de vértices;
Inicializa todos os valores de cores[ ] com 1;
Enquanto resultado = falso {
        Cria todas as permutações para cores [ ];
        Atribui cores dos vértices, de acordo com valores de cores [ ];
        Verifica se solução é válida{
                Passa-se por cada vértice comparando suas cores com os vértices adjacentes;
                Se a solução proposta for válida, resultado = verdadeiro;
                Se for não for válida, resultado = falso;
        }
        Se solução não é válida, Verifica se pode acrescentar um novo valor para cores[]
        Se sim, acrescenta-se um valor ao espaço solução de cores[]
        contagem_rodadas = novo valor;
        /* Se o vetor antes era cores = {1, 1, 1, 1} acrescenta-se um valor e cores = {1, 1, 1, 2}
        Comportamento se repete até cores= {1, 2, 2, 2} depois se torna cores = {1, 2, 3, 3} até
        que
        termina com o espaço máximo que é cores = {1, 2, 3, 4} para um grafo com 4 vértices
        Para vértices com mais cores resultado é cores = {1, 2, 3, ... número de vértices} */
}
Imprime-se vetor cores[] no arquivo alocacao.txt;
Imprime-se a variável contagem_rodadas no arquivo rodada.txt;
Desaloca grafo criado;
```

**Código 1.** Pseudocódigo do algoritmo de força bruta. Implementação em C está no arquivo *main_fb.c*

Como o algoritmo utilizado se encerra ao encontrar a primeira solução válida, o resultado impresso no arquivo *alocacao.txt* pode ser um pouco diferente dos resultados fornecidos nos casos de teste para força bruta. Isso ocorre porque o algoritmo testou primeiro uma permutação que é diferente da resposta. Um exemplo é o resultado para o arquivo *entrada1.txt.*

| Resultados para o arquivo *entrada1.txt* | |
|---|---|
| Solução deste trabalho | Solução dos arquivos de teste |
| 1 2 | 1 1 |

| | |
|---|---|
| 2 1 | 2 2 |
| 3 1 | 3 3 |

**Tabela 1.** Comparação entre resultados de força bruta

Ambas as respostas são soluções válidas, porém com as cores dos vértices invertidas pois se trata de uma permutação da solução.

A solução de força bruta possui um péssimo desempenho, pois conseguiu encontrar a solução até o arquivo de teste *entrada12.txt.* A execução do arquivo *entrada13.txt* foi interrompida quando o programa atingiu 2 horas e 21 minutos de execução e não havia previsão de quando a solução seria encontrada.

## 2.2 Solução do problema usando heurística

Como o problema de coloração em grafo é conhecido como NP-Completo não é possível encontrar uma solução ótima em tempo polinomial, por isso utilizamos de alguma estratégia para encontrar uma solução que resolva o problema em um tempo menor.

A heurística adotada foi utilizar o método de *backtracking* para encontrar uma solução. A ideia é atribuir uma cor a cada vértice, começando do vértice inicial 1. Antes de atribuir uma cor, verifica-se se não existe a mesma cor nos vértices adjacentes. Caso seja seguro, segue-se com a cor proposta, caso não seja, dá-se um passo para trás, *backtracking*, e tenta-se uma nova cor. Este processo se repete até que o grafo esteja completamente colorido.

```
Ler grafo de entrada;
Cria grafo usando lista de adjacências;
Cria vetor de cores[ ] de 0 até N = número de vértices;
Inicializa todos os valores de cores[ ] com 0;

Colorir_grafo (int v, int color[ ], grafo) {
        /* Condição para verificar se chegou ao caso base da recursão  */
        Se ( vértice v == num_vertices )
                retorna verdadeiro;
        Para ( c = 1 ; c < num_vertices ; c++ )
                Se ( Verificar_cor == verdadeiro )  // Pseudocódigo de Verificar_cor ao final
                {
                        Colore vértice v com cor c;
                        Adicionar c ao vetor solução cores[ ];
                        Se ( c > contagem_rodadas )
```

```
                                contagem_rodadas = c;

                            /* Chama função para o próximo vértice v+1  */
                            Se ( Colorir_grafo( v+1, color[ ], grafo) == verdadeiro )
                                retorna verdadeiro;

                            /* Se a cor c não levar a solução, então ela é removida */
                            cores[v] = 0;
                            Remover cor do vértice v;
                    }

            /* Se a solução não for encontrada, retorna falso  */
            retorna falso;
}
Imprime-se vetor cores[] no arquivo alocacao.txt;
Imprime-se a variável contagem_rodadas no arquivo rodada.txt;
Desaloca grafo criado;

/* Função que verifica se uma cor c pode ser atribuída a um vértice v   */
Verificar_cor ( int v, int c, int cores[ ], grafo){

            /* Analisa se o vértice atual já possui a cor   */
            Se ( v.cor && c == cores[v] )
                    retorna falso;

            /* Verifica se os vértices adjacentes de v possuem a mesma cor   */
            Enquanto ( != NULL ){
                    Se ( v->adjacente.cor && c == cores[v->adjacente] )
                            retorna falso;
            }

            /* Caso essa cor seja segura, retorna verdadeiro   */
            retorna verdadeiro;
}
```

**Código 2.** Pseudocódigo do algoritmo de backtracking. Implementação em C está no arquivo *main_h.c*

A heurística não encontrou o mesmo resultado que alguns casos de testes propostos, pois realizou um número de rodadas um pouco acima do resultado. Porém, seu tempo de execução é muito menor que o algoritmo de força bruta, favorecendo seu uso.

# ANÁLISE TEÓRICA

## 3.1 Análise teórica do custo assintótico de tempo - Força bruta

Podemos realizar a análise de custo assintótico avaliando o número de candidatas a solução que são geradas pelo algoritmo, além da verificação da mesma. Para verificar se uma solução candidata é válida é preciso percorrer todo o grafo analisando se para cada vértice $V$ seus vértices adjacentes possuem a mesma cor. Esta operação possui um custo de $O(V+E)$, onde $V$ é o número de vértices e $E$ é o número de arestas.

No algoritmo de força bruta são criadas $V!$ candidatas a solução, pois para um grafo com $V$ vértices, o máximo de cores que se pode colori-lo é $V$, então $V!$ é o número de permutações em um vetor com $V$ posições. Na implementação realizada, para cada uma dessas $V!$ permutações em um vetor de cores, existem $n$ vetores com valores diferentes entre si que devem ser testados para encontrar uma solução.

Por exemplo, para um grafo com 4 vértices, $V=4$, começamos a testar as soluções com um vetor de cores $C = \{1, 1, 1, 1\}$. A partir deste vetor $C$, faz-se $4!$ permutações, onde cada solução é testadas com custo $O(V+E)$. Como a solução não é encontrada, soma-se um valor a uma das posições e $C$ torna-se $C = \{1, 1, 1, 2\}$. Repetem-se as $V!$ permutações e verificações. Em uma outra interação têm-se $C = \{1, 1, 2, 2\}$, assim por diante, até chegar na última interação em que $C = \{1, 2, 3, 4\}$. Estas alterações no valor do vetor $C$ ocorrem $n$ vezes. Portanto, ao final do algoritmo de força bruta, têm-se uma complexidade de $O(nV!)$

.

## 3.2 Análise teórica do custo assintótico de tempo - Heurística

Podemos realizar a análise de custo assintótico avaliando o número de comparações que são realizadas pelo algoritmo para determinar se um vértice será colorido, ou não. Para realizar esta operação, são realizadas $O(CV)$ comparações,

onde $C$ é o número de cores até o momento e $V$ o número de vértices. Ao final têm-se que a complexidade de tempo que no pior caso é $O(VC^V)$.

### 3.3 Análise Teórica do Custo Assintótico de Espaço

Para fazer a análise de custo de espaço, foca-se na forma em que o grafo é implementado. Como ambas as soluções foram feitas utilizando lista de adjacências e um vetor de cores de tamanho $V$, para armazenar a solução, conclui-se que a complexidade de espaço é $O(2V + E)$.

# ANÁLISE EXPERIMENTAL

A máquina utilizada para rodar os testes possui 6,0 GB de memória RAM, com processador Core™ i7-5500U CPU 2.40Hz, com sistema operacional *Linux Ubuntu.*

### 4.1 Comparação de tempo entre as soluções propostas

Os testes abaixo foram realizados com os casos de testes disponibilizados no arquivo *exemplos.zip* que foram fornecidos junto ao enunciado deste trabalho prático.

| ARQUIVO TESTE | NÚMERO DE RODADAS HEURÍSTICA | TEMPO HEURÍSTICA | NÚMERO DE RODADAS FORÇA BRUTA | TEMPO FORÇA BRUTA |
|---|---|---|---|---|
| entrada1.txt | 2 | 0,0050 | 2 | 0,0060 |
| entrada2.txt | 3 | 0,0050 | 3 | 0,0030 |
| entrada3.txt | 2 | 0,0030 | 2 | 0,0020 |
| entrada4.txt | 4 | 0,0040 | 4 | 0,0020 |
| entrada5.txt | 3 | 0,0030 | 3 | 0,0060 |
| entrada6.txt | 3 | 0,0040 | 3 | 0,0060 |
| entrada7.txt | 2 | 0,0040 | 2 | 0,0020 |
| entrada8.txt | 3 | 0,0040 | 3 | 0,0100 |
| entrada9.txt | 7 | 0,0050 | 7 | 0,0450 |

| | | | | |
|---|---|---|---|---|
| entrada10.txt | 8 | 0,0050 | 8 | 0,3050 |
| entrada11.txt | 9 | 0,0040 | 9 | 3,5020 |
| entrada12.txt | 10 | 0,0040 | 10 | 45,7220 |
| entrada13.txt | 20 | 0,0040 | 20 | 7.980,0000* |

**Tabela 12:** Nome dos arquivos de testes contendo: número rodadas (cores) encontradas em cada solução e respectivo tempo. *Execução foi interrompida por chegar em 2h20min de execução e ainda não encontrar uma solução.

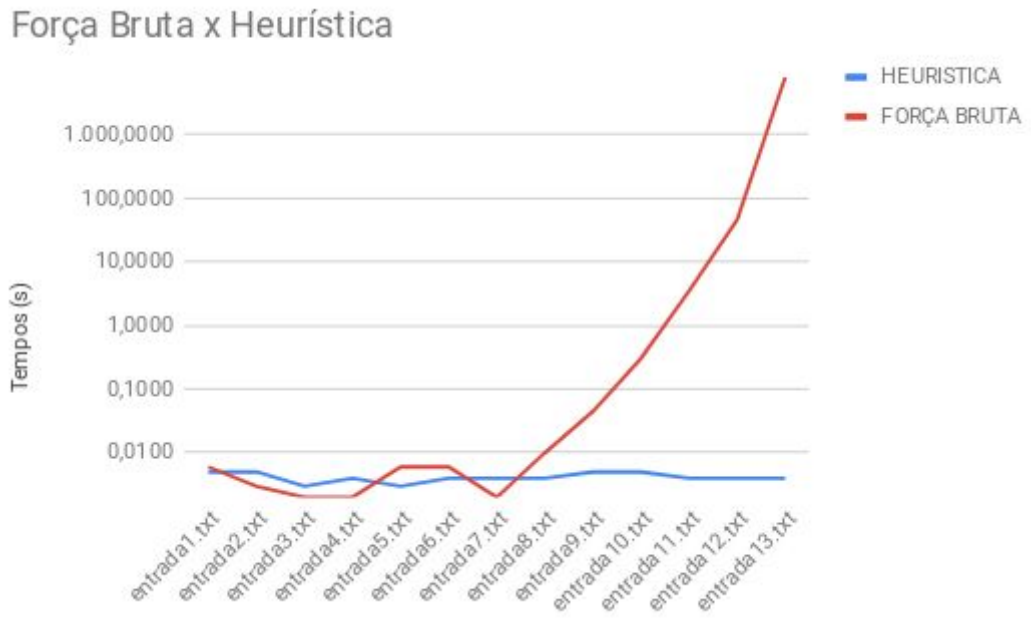


**Gráfico 1:** Comparação de tempo entre as soluções de força bruta e heurística. Eixo de tempo vertical em escala logarítmica.

O gráfico demonstra que o tempo de execução do algoritmo é fatorial. Tal comportamento comprova a análise teórica do item 3 desta documentação.

## 4.2 Comparação entre heurística e solução ótima

Os testes abaixo foram realizados com os casos de testes disponibilizados no arquivo *TesteHeuristica.zip* que foram fornecidos junto ao enunciado deste trabalho prático.

| ARQUIVO TESTE | RODADAS ALGORITMO ÓTIMO | RODADAS HEURÍSTICA | % ERRO NO TESTE | TEMPO DE EXECUÇÃO HEURÍSTICA (s) |
|---|---|---|---|---|
| entrada1.txt | 4 | 4 | 0,00% | 0,0040 |
| entrada2.txt | 5 | 5 | 0,00% | 0,0040 |
| entrada3.txt | 6 | 6 | 0,00% | 0,0330 |

| | | | | |
|---|---|---|---|---|
| entrada4.txt | 8 | 8 | 0,00% | 0,0060 |
| entrada5.txt | 10 | 16 | 60,00% | 0,0050 |
| entrada6.txt | 13 | 21 | 61,54% | 0,0070 |
| entrada7.txt | 30 | 30 | 0,00% | 0,0100 |
| entrada8.txt | 5 | 18 | 260,00% | 0,0250 |
| entrada9.txt | 31 | 31 | 0,00% | 0,0290 |
| entrada10.txt | 54 | 54 | 0,00% | 0,0310 |
| **MÉDIA DE ERRO DA SOLUÇÃO:** | | | 38,15% | |

**Tabela 3:** Nome dos arquivos de testes contendo: rodadas(cores) encontradas por um algoritmo ótimo, rodadas encontradas pela heurística, a porcentagem de erro entre as soluções e o tempo de execução do algoritmo e a média de erro da solução heurística.

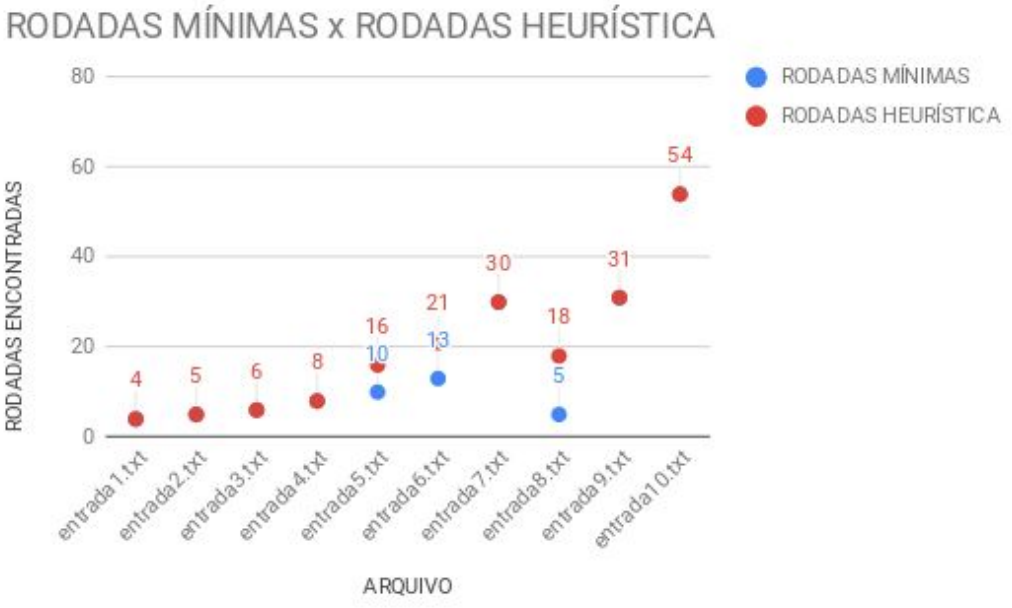


**Gráfico 2:** Comparação de rodadas encontradas entre a solução ótima e heurística.

A análise demonstra que a heurística apresentou uma média de eficiência de 61,85% em comparação com a solução ótima. Este é um bom resultado visto que o problema é NP-Completo o que torna inviável encontrar a solução ótima.

# CONCLUSÃO

Neste trabalho, foi resolvido o problema de atualização de servidores da empresa *BH Software*. O problema foi resolvido utilizando duas abordagens

distintas: força bruta e heurística. A análise de complexidade teórica de tempo foi comprovada através de experimentos que utilizaram entradas grandes suficientes para analisar o comportamento assintótico. Ao final, podemos concluir que devido ao problema de coloração em grafo se tratar de um problema NP-Completo uma solução heurística é a melhor escolha, visto que não é possível encontrar uma solução ótima em tempo polinomial.

# REFERÊNCIAS


USP, O problema de coloração em grafos e NP completude: http://conteudo.icmc.usp.br/pessoas/andretta/ensino/aulas/sme0216-5826-2-15/grupo5.pdf

Geeks for geeks coloração usando o algoritmo backtracking em matriz de adjacências: https://www.geeksforgeeks.org/backttracking-set-5-m-coloring-problem/

Tutorials point, Back Tracking Graph Coloring Problem: https://www.tutorialspoint.com/analysis_of_algorithm/back_tracking_graph_coloring_problem.asp

Wikipedia, Graph Coloring Problem: https://en.wikipedia.org/wiki/Graph_coloring