# Introdução à Otimização Combinatória Aplicada

**Terceiro Trabalho Prático**
**Entrega Sugerida: 19/07/2020**

## 1 Introdução

No terceiro Trabalho Prático (TP03) será solicitada a entrega de um arquivo **.zip** contendo um programa que solucione o problema apresentado na próxima seção, bem como, para cada instância, um arquivo **.sol** contendo a solução que seu programa encontrou para a mesma. Por conta de detalhes do corretor, no **.zip** também devem estar os arquivos **Makefile** e **verifier.py**, que são disponibilizados junto do trabalho. Além disso, os itens a seguir devem ser respeitados:

- O TP03 deve ser feito individualmente e plágio não será tolerado;
- O TP03 deve ser entregue no run codes (https://run.codes);
- Seu programa deve apresentar um código em uma das seguintes linguagens (C, C++, Java, Python 3) e deve conter um cabeçalho com informações do estudante, como: nome, curso (caso se aplique) e RA (ou RG, se for membro externo);
- Cada estudante deve se cadastrar no run codes (https://run.codes) informando Nome Completo, escolhendo "UFSCar - Universidade Federal de São Carlos" no campo Universidade e colocando seu RA (ou RG, se for membro externo) no campo Núm. Matrícula. Depois de cadastrado, basta logar no run codes e se matricular na disciplina "IOCA - Introdução à Otimização Combinatória Aplicada" usando o Código de Matrícula EZMQ.
- Caso utilize PL ou PLI, pode usar resolvedores como OR-Tools, CPLEX e Gurobi.

## 2 Coloração de Grafos

Neste trabalho, seu objetivo é projetar um algoritmo que encontre a coloração mínima de um grafo. Você receberá um grafo e sua tarefa será rotular seus vértices com a menor quantidade possível de cores, de modo que vértices vizinhos não tenham a mesma cor. Veja a Figura 1 para um exemplo. Note que, a coloração mostrada não é a mínima para o exemplo, visto que é possível utilizar apenas duas cores.

O problema pode ser matematicamente formulado da seguinte forma. Dado um grafo $G = (V, E)$, com vértices $V = \{0, \ldots, |V| - 1\}$ e arestas $E$, seja $c_i \in \mathbb{N}$ a variável denotando a cor do vértice $i$. Assim, o objetivo é minimizar o maior valor $c_i$ ($max_{i \in V}\ c_i$), sendo que $c_i \neq c_j$ para toda aresta $(i, j) \in E$ do grafo.

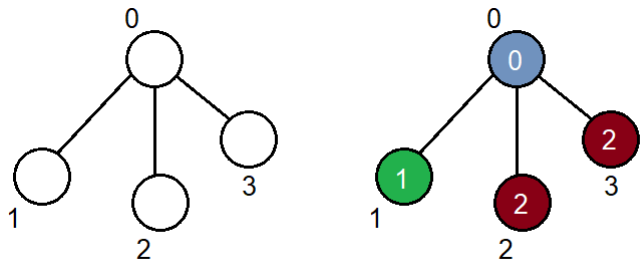


Figura 1: À esquerda temos um grafo e à direita um exemplo de coloração para o mesmo. Os vértices do grafo são identificados pelos números em preto, enquanto as cores são rotuladas pelos números em branco.

## Entrada:

Cada instância contém um único caso de teste. A primeira linha contém dois inteiros: o número de vértices $|V|$ e o número de arestas $|E|$. As próximas $|E|$ linhas contêm as informações das arestas. Especificamente, cada linha possui 2 números, $u$ e $v$, com $u, v \in \{0, \ldots, |V| - 1\}$, indicando que existe uma aresta $(u, v)$ no grafo.

## Saída:

A saída de seu programa deve ser composta por duas linhas. A primeira contendo $obj$, a quantidade de cores usadas na coloração (i.e, o valor objetivo). A próxima linha é uma lista de $|V|$ valores, um para cada variável $c_i$, correspondendo à qual cor o vértice $i$ está associado.

## Exemplo de entrada

No exemplo abaixo temos um grafo com quatro vértices {0, 1, 2, 3} e três arestas, sendo que $E = \{(0, 1), (1, 2), (1, 3)\}$

```
4 3
0 1
1 2
1 3
```

## Saída possível para esse exemplo

Neste resultado estamos usando três cores para a coloração. O vértice 0 está associado à cor 0, o vértice 1 à cor 1, e os vértices 2 e 3 à cor 2.

```
3
0 1 2 2
```

# 3 Instruções

Disponibilizamos o trabalho no arquivo **tp03.zip**. Este contém o arquivo **solver.py** que realiza a leitura de uma instância, executa um algoritmo básico para o problema, devolve uma solução no formato de saída descrito acima, e grava essa solução no arquivo <**nome_da_instancia**>**.sol** na mesma pasta da instância lida. Modifique a função **ColoringNaive()** do **solver.py** para resolver o problema proposto. Sua implementação pode ser testada com o comando

```
python ./solver.py ./data/<nome_da_instancia>
```

Você pode escolher entre implementar sua solução diretamente em python, modificar a função para chamar uma aplicação externa, ou desenvolver um código na linguagem que desejar, que substitua o **solver.py**.

As instâncias que devem ser resolvidas também estão no **tp03.zip**, mais especificamente na pasta **./data/**. Também disponibilizamos algumas instâncias menores na pasta raiz, cujas soluções não serão cobradas, mas que podem ser úteis nos seus testes durante a implementação.

Sua submissão deve corresponder a um arquivo **.zip** que contém, além do seu programa, um arquivo <**nome_da_instancia**>**.sol** para cada instância. O conteúdo desses deve ser a solução encontrada por seu programa ao resolver a instância correspondente. Atenção para colocar os arquivos na raiz do **.zip**, e não dentro de uma pasta. No **.zip** também devem estar os arquivos **Makefile** e **verifier.py**, que estão disponíveis na pasta **./sols/** do **tp03.zip**. Em particular, vocês podem usar o programa **verifier.py**, que valida as soluções encontradas, para facilitar seus testes durante a implementação.

## Cálculo da Nota

Soluções inviáveis (i.e., aquelas que não seguem o formato de saída ou violam as restrições do problema) irão receber 0 pontos. Soluções viáveis irão receber pelo menos 3.33 pontos. Soluções viáveis superando um primeiro padrão de qualidade irão receber 6.66 pontos, e soluções que superam um alto padrão de qualidade irão receber 10 pontos. No run codes existem testes distintos, com a mesma instância, para verificar esses diferentes critérios. Para ser aprovado, o estudante deve obter pelo menos 70% dos pontos do trabalho.