Universidade Federal de Juiz de Fora
DCC059 - Teoria dos Grafos
**Especificação do Trabalho 2**

# Motivação

Durante a disciplina de grafos, foram apresentados os principais conceitos e algoritmos de Teoria dos Grafos. Os algoritmos estudados em sala (Prim, Kruskal, Dijkstra, Floyd, por exemplo) são amplamente conhecidos por serem capazes de fornecer, de forma eficiente, uma solução exata para os respectivos problemas.

Entretanto, muitos problemas envolvendo grafos são problemas de Otimização Combinatória considerados intratáveis, isto é, problemas para os quais não são conhecidos algoritmos capazes de encontrar solução ótima em tempo polinominal. Estes problemas são frequentemente resolvidos por algoritmos heurísticos, que têm como objetivo encontrar boas soluções de forma rápida.

Neste trabalho, os alunos deverão desenvolver um projeto de programação relacionado a um problema intratável de Otimização Combinatória. Além de colocar em prática os conhecimentos adquiridos na disciplina, os alunos precisarão organizar, comparar e analisar os resultados obtidos para diferentes instâncias do problema, fornecendo um relatório detalhado do projeto.

# Objetivos

O objetivo geral deste trabalho é que cada grupo de alunos desenvolva um projeto de implementação para resolver heuristicamente um problema clássico de grafos. Espera-se que os alunos, além da implementação, trabalhem na execução e comparação de experimentos, análise de resultados de algoritmos e na confecção de um relatório sobre o projeto.

# Desenvolvimento do trabalho

O trabalho requer uma implementação, um relatório de desenvolvimento e a apresentação do trabalho. Cada um destes itens é detalhado a seguir.

## Projeto de implementação

O projeto de implementação deverá ser desenvolvido **na linguagem C++** e deverá incluir:

- um tipo abstrato de dados Grafo, armazenando nós e arestas/arcos em uma estrutura de dados de sua escolha, de forma a assegurar que represente grafos conforme sejam definidas as instâncias do problema (direcionados/não direcionados, ponderados nas arestas, ponderados nos vértices, etc);
- geração de semente de randomização baseada em data/hora (chamada uma única vez em todo o código) e impressão desta para que seja possível repetir um teste específico, caso necessário (também é interessante permitir que a semente seja informada como um parâmetro de execução);

- leitura dos dados de arquivo, em formato padrão das instâncias do problema na literatura;
- um algoritmo guloso para o problema;
- um algoritmo guloso randomizado, em que o valor $\alpha$ e o número de iterações sejam parâmetros;
- um algorimo guloso randomizado reativo, em que os valores de $\alpha$, o número de iterações e o tamanho dos blocos sejam parâmetros;
- impressão da solução (em arquivo ou na tela) preferencialmente em formato que facilite sua visualização (permitindo copiar e colar em http://csacademy.com/app/graph$_e$*ditor*/, por exemplo).

Para facilitar a organização dos resultados obtidos, sugere-se que, no final da execução de cada algoritmo, o código acrescente uma linha em um arquivo '.csv' contendo os dados da execução. Neste caso, o ideal é que os dados incluam:

- data/hora do teste,
- instância,
- eventuais parâmetros do teste, dependendo do problema (capacidades, limites máximos, cardinalidade máxima, etc),
- algoritmo executado no teste,
- parâmetros do algoritmo ($\alpha$, número de iterações, tamanho do bloco),
- semente de randomização,
- tempo gasto na execução,
- valor da melhor solução alcançada.

Embora nem todos os algoritmos apresentem todos esses dados, construir uma tabela única (onde algumas colunas podem estar vazias nas linhas correspondentes ao guloso, por exemplo) ajuda bastante na hora de consolidar os resultados. Além dos dados listados, o $\alpha$ que forneceu a melhor solução do algoritmo reativo e a média dos valores das soluções obtidas em todas as iterações dos algoritmos randomizados, por exemplo, também podem gerar resultados interessantes no final.

Tanto a corretude da implementação quanto a organização e clareza do código serão avaliadas. Estejam à vontade para criar classes auxiliares e funções adicionais que contribuam para um código bem estruturado, organizado e de fácil compreensão. O código deve estar compilando e executando (se não compilar, a nota é zero).

Será incluída uma atividade no Classroom para que os alunos incluam o link do git onde o código estará disponível. No diretório raiz do git, insira um arquivo '.txt' com uma descrição mínima de como executar o programa via linha de comando (em ambiente Linux), informando como executar cada algoritmo (linha de comando e parâmetros).

## Experimentos e Relatório

Para os experimentos, cada grupo pode escolher três valores de alfa, de acordo com o que for mais adequado para o seu algoritmo. Os valores apresentados no documento são apenas exemplos ilustrativos.

Os algoritmos devem ser executados 10 vezes para cada instância (cada vez com uma semente diferente), registrando ao menos a melhor solução e o tempo médio de processamento (em segundos). No algoritmo randomizado, o algoritmo construtivo deve ser chamado pelo menos 30 vezes. No algoritmo randomizado reativo, o construtivo deve ser chamado pelo menos 300 vezes, com blocos de 30 a 50 iterações para possibilitar a atualização das probabilidades.

No relatório, devem ser inseridas no mínimo as seguintes tabelas de resultados:

- Desvios percentuais relativos às melhores soluções obtidas em cada algoritmo para cada instância (comparadas à melhor solução conhecida da respectiva instância). Observe que este resultado corresponde ao melhor resultado alcançado em uma das 10 execuções.
- Desvios percentuais relativos à média da melhor solução obtida em cada execução do algoritmo para cada instância (comparadas à melhor solução conhecida da respectiva instância). Aqui, será utilizada a média do resultado das 10 execuções.
- O tempo médio das 10 execuções de cada algoritmo.

No Classroom, foi disponibilizada uma planilha para ajudar no cálculo das tabelas de desvio percentual. Preenchendo os dados obtidos, a tabela calcula os desvios percentuais e formata as linhas para colar no latex.

O relatório do trabalho deverá seguir o modelo disponibilizado no Classroom. O referido modelo especifica o que deve ser fazer parte do relatório, incluindo exemplos de como algoritmos e tabelas devem ser inseridos. O relatório deve conter até 10 páginas (sem contabilizar capa, sumário e bibliografia) e **NÃO** deve conter código fonte.

## Apresentação do trabalho

A apresentação do trabalho será agendada em data posterior à entrega do trabalho. Todos os integrantes deverão estar presentes na apresentação e deverão conhecer todos os aspectos do trabalho. Por este motivo, recomenda-se que cada aluno participe de todas as etapas: implementação, testes, análise de resultados e escrita do relatório. As notas serão individuais, de acordo com o acompanhamento e a participação individual de cada aluno.

A apresentação de trabalho será realizada em laboratório, sendo que o código deverá ser executado em uma das máquinas do laboratório no momento da apresentação.

# Temas

Cada trabalho deve se concentrar em um dos temas especificados abaixo. Os alunos deverão indicar em formulário sua preferência, mas a atribuição final será realizada pela professora e disponibilizada no Classroom.

## 1. $L(p, q)$-coloring

**Definição.** Dado um grafo simples e não direcionado $G = (V, E)$ e dois inteiros não negativos $p$ e $q$, o problema de $L(p, q)$-*coloring* consiste em encontrar uma coloração $f : V \to Z^+$ dos vértices tal que:

- se $u$ e $v$ são vértices adjacentes (distância 1), então $|f(u) - f(v)| \geq p$;
- se $u$ e $v$ estão a distância 2, então $|f(u) - f(v)| \geq q$.

Neste problema de coloração, cada cor corresponde portanto a um número inteiro e o objetivo é minimizar a maior cor utilizada na coloração.

**Instâncias de teste.** Os conjuntos de instâncias podem ser obtidos em:

`https://cedric.cnam.fr/~porumbed/graphs/`

## 2. Defective Coloring

**Definição.** Dado um grafo $G = (V, E)$ e um inteiro $d$, uma *coloração $d$-defectiva* permite que vértices adjacentes recebam a mesma cor, desde que, para cada vértice, o número de vizinhos com a mesma cor não exceda $d$. Isto é, neste problema, cada vértice $v \in V$ pode ter no máximo $d$ vizinhos com a mesma cor que foi atribuída a $v$.

O objetivo é minimizar o número de cores utilizadas.

**Instâncias de teste.** As instâncias podem ser obtidas em:

`https://cedric.cnam.fr/~porumbed/graphs/`

## 3. Degree-Constrained Minimum Spanning Tree (DC-MST)

**Definição.** Dado um grafo conexo e ponderado $G = (V, E, w)$ e um limite de grau máximo $d$ para os vértices, o problema da *Árvore Geradora Mínima com Restrição de Grau* consiste em encontrar uma árvore geradora $T \subseteq E$ tal que:

- $T$ conecta todos os vértices de $V$;
- o grau de $v$ em $T$ seja no máximo $d$, para todo $v \in V$;
- o custo total $\sum_{e \in T} w(e)$ seja mínimo.

**Instâncias de teste.** Instâncias de referência podem ser obtidas na biblioteca DC-MST:

`https://andreas-ernst.github.io/Mathprog-ORlib/info/readmeDCMST.html`

## 4. Capacitated Minimum Spanning Tree (CMST)

**Definição.** Dado um grafo conexo e ponderado $G = (V, E, w)$, um vértice raiz $r \in V$ e uma capacidade $C$, o problema da *Capacitated Minimum Spanning Tree* consiste em encontrar uma árvore geradora enraizada em $r$ tal que:

1. cada vértice pertença a uma subárvore conectada diretamente a $r$;
2. a soma das demandas (ou pesos) dos vértices em cada subárvore não exceda a capacidade $C$;
3. o custo total das arestas da árvore seja mínimo.

**Instâncias de teste.** Instâncias clássicas podem ser obtidas em:

`https://people.brunel.ac.uk/~mastjjb/jeb/orlib/capmstinfo.html`

(utilizar o arquivo `capmstnew.zip` disponível na página).

# Dúvidas

Cabe a cada grupo avaliar o tempo que demanda o experimento de forma a assegurar que as entregas sejam feitas até a data estabelecida.

O trabalho é em grupo e deve ser feito sempre segundo as orientações. Leve à professora as decisões de projeto para serem discutidas, sempre que necessário. Em caso de dúvidas, enviem mensagem via Classroom.