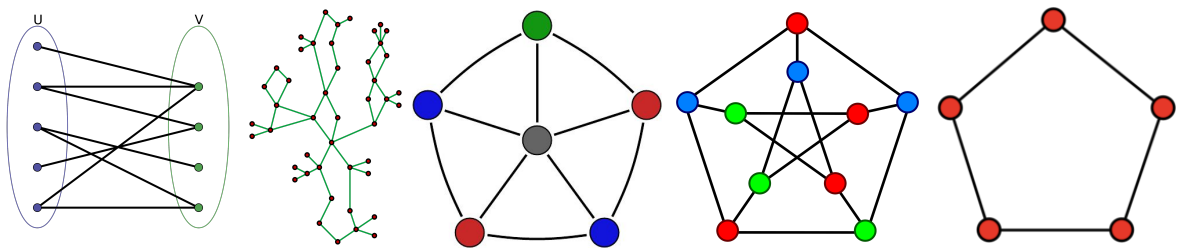
**Trabalho 3 (Coloração de Grafos) - Teoria dos Grafos**
**Grupo: Lucas Rolim  e Anderson Barbosa**

# Decisões de implementação

Julgamos que, no contexto do trabalho, a melhor opção seria utilizar um algoritmo guloso. Com isso em mente, após procurar por diversas heurísticas e suas eficiências, escolhemos por implementar uma versão do famoso algoritmo de coloração DSatur. O algoritmo implementado tem sua gula na saturação das arestas, ou seja, escolhe como próximo vértice a color sempre aquele o qual possui o maior número de vizinhos coloridos de cores diferentes.

Um dos principais motivos para a escolha foi que além de obter quase sempre uma boa estimativa do número mínimo de cores a abordagem DSatur também garante uma solução ótima para diversos tipos de grafos. Como pode ser visto no artigo científico *"The smallest hard-to-color graph for algorithm DSATUR" (R. Janczewski , M. Kubale, K. Manuszewski, K. Piwakowski)* a solução ótima é garantida nos seguintes casos:

- Grafo bipartido
- Grafo em forma de roda
- Grafo multipartido completo
- Grafo em forma de cacto
- Grafo em forma de colar




Para a implementação de tal algoritmo criamos dois novos métodos na classe e também um vetor de cores para armazenar a coloração dos vértices do grafo.

O primeiro método criado foi o *"getMostRestrictedVerte"*, um método com o objetivo de retornar sempre o vértice com o maior número de vizinhos que já estão coloridos de cor diferente. Para a implementação desse método basicamente utilizamos um vetor de inteiros para armazenar e comparar temporariamente as cores já utilizadas para colorir os vizinhos e quantas essas cores são. No fina de sua execução esse método retorna o vértice desejado para a heurística.

O segundo método principal é o *"colorGraph"*, método este que implementa de fato a lógica do algoritmo guloso mencionado.  O objetivo dessa função é selecionar o vértice retornado pelo método *"getMostRestrictedVerte"* e colorir o seus vizinhos sempre com a menor cor possível. Essa coloração é salva no vetor de inteiros colors, que é global à classe.  A cada interação também é atualizada a variável *"CNumber"*, que armazena o número de cores utilizados até o momento para colorir o grafo.

Por fim, também foram implementados alguns três métodos auxiliares. O primeiro deles, “validColoring”, serve para checar se a coloração do grafo é válida através de uma comparação da cor de cada vértice com as cores de seus vizinhos. O segundo, *“getMaxDegreeVertex”*, tem objetivo de auxiliar a heurística do DSatur em caso de “empate” de vértices. E, por último, o método *“probColor”* permite ao usuário testar a coloração mínima partindo diversos vértices de início diferentes *(uma espécie de força bruta controlada)*.

# Estudo de caso

**Os resultados obtidos foram:**

| Grafo | Vértices | Arestas | Cores | Tempo (s) |
|---|---|---|---|---|
| G1.txt | 450 | 5735 | 13 | 0.459789 |
| G2.txt | 450 | 9804 | 16 | 1.07749 |
| G3.txt | 450 | 8170 | 20 | 1.20282 |
| G4.txt | 450 | 16681 | 30 | 2.55503 |
| G5.txt | 450 | 8264 | 27 | 1.07184 |
| G6.txt | 450 | 17344 | 35 | 2.90696 |
| G7.txt | 10 | 26 | 2 | 4.9e-05 |
| G8.txt | 10 | 13 | 3 | 2.4e-05 |
| G9.txt | 100 | 1001 | 2 | 0.012738 |
| G10.txt | 100 | 351 | 5 | 0.004421 |
| G11.txt | 100 | 2001 | 2 | 0.020289 |
| G12.txt | 100 | 2166 | 19 | 0.085895 |
| G13.txt | 1000 | 5001 | 8 | 0.748808 |
| G14.txt | 1000 | 5566 | 14 | 1.03591 |
| G15.txt | 1000 | 20001 | 8 | 4.62737 |
| G16.txt | 1000 | 21226 | 22 | 7.03328 |
| G17.txt | 10000 | 87366 | 30 | 314.693 |
| G18.txt | 100000 | 282559 | 41 | 1369.77 |
| G19.txt | 10000 | 756 | 7 | 0.00622 |

| | | | | |
|---|---|---|---|---|
| G20.txt | 191 | 2361 | 8 | 0.036962 |

Nenhum código estava disponível no site para que fosse possível fazer o teste da coloração. Visto isso, para realizar tais testes. se seguinte função foi utilizada:

```
bool validColoring() {
  for (int i=1; i<=verticesNumber; i++) {
     for (int j=1; j<=container.getDegree(i); j++) {
        int x=container.getNeighbor(i,j);
        if (color[i-1]==color[x-1]) {
           return 0;
        }
     }
  }
  return 1;
}
```