Trabalho Prático 2 - Essa coloração é gulosa?

Descrição do problema

Problemas de coloração em grafos possuem uma vasta área de aplicações, pois conseguem codificar muitas propriedades entre entidades. Existem inúmeras variações de coloração em grafos e uma das mais comuns é a coloração de vértices. Dado um grafo G e um conjunto de cores C, o objetivo é colorir os vértices de G de forma que, se dois vértices forem adjacentes, eles devem possuir cores diferentes. O problema clássico envolve tentar encontrar o menor conjunto C de cores que permite colorir os vértices de G seguindo essa regra. Existem várias estratégias para tentar atingir esse objetivo, mas as grandes mentes da computação chegaram a conclusão através de um arcabouço extremamente engenhoso de que esse problema é **difícil** (e você vai ouvir falar muito sobre isso se cursar algoritmos I e II). O nosso problema será um pouco diferente, você irá receber um grafo e uma coloração de seus vértices e deve responder a seguinte pergunta: "É possível obter essa coloração através de um **algoritmo guloso**?"

Para entender melhor o problema, vamos formalizar um pouco as colorações e definir o que será nosso algoritmo guloso de coloração.

Colorações de vértices em grafos

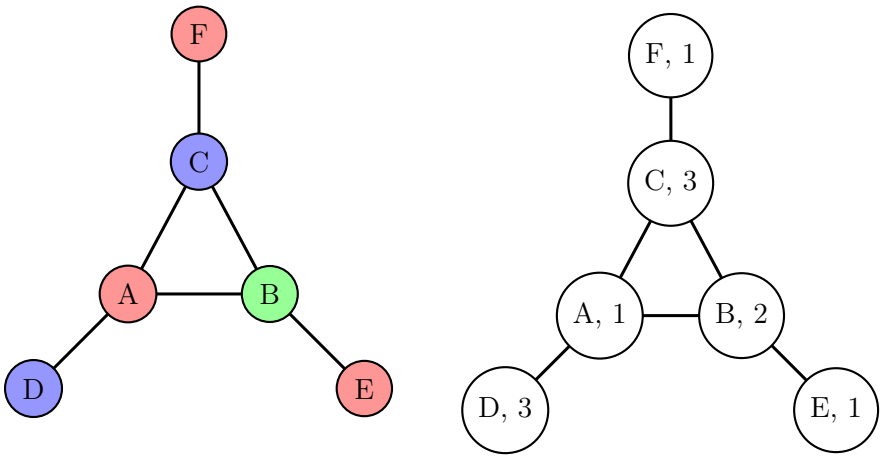
Seja $G = (V, E)$ um grafo e um inteiro positivo k, uma $k-$coloração de G é uma função c que associa os vértices de G com inteiros de 1 até k. Mais formalmente:

$$c : V(G) \to \{1, 2, \ldots, k\}$$

Uma $k-$coloração de G é **própria** quando vértices adjacentes possuem cores diferentes, ou seja:

$$\forall u, v \in V(G) : uv \in E(G) \to c(u) \neq c(v)$$

Se G admite uma $k-$coloração própria, então dizemos que G é $k-$colorível.



(a) Grafo colorido utilizando cores

(b) Mesmo grafo colorido utilizando números para representar as cores

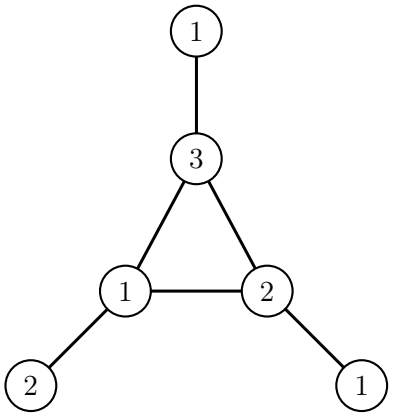
Figura 1: Ao invés de utilizar cores em si, utilizamos números para representar as cores. Em alguns exemplos o rótulo dos vértices também será omitido.

Colorações gulosas

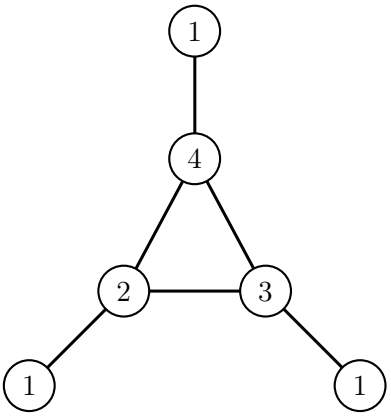
Existem várias formas de se definir uma coloração gulosa. Neste trabalho uma coloração é gulosa quando ela pode ser obtida através do seguinte algoritmo:

1. Escolha uma permutação P dos vértices de G, ou seja, um ordenação total dos vértices.
2. Para cada vértice v da permutação P, tomado na ordem de P, atribua a ele a menor cor disponível. Uma cor c é dita disponível se $c \in C$ e nenhum vizinho de v possui cor c. Caso não haja cor disponível, "crie"uma cor nova acrescentando mais uma cor a C e atribua a nova cor a v.

Baseado no segundo passo do algoritmo, podemos imaginar que nem sempre o algoritmo guloso irá utilizar a mesma quantidade de cores. A Figura 2 ilustra um exemplo de que a quantidade de cores utilizadas pode variar de acordo a permutação escolhida.



(a) Coloração gulosa obtida no grafo da Figura 1a com a permutação ABCDEF



(b) Coloração gulosa obtida no grafo da Figura 1a com a permutação DFEABC

Figura 2: O número de cores utilizadas em colorações gulosas em um mesmo pode variar de acordo com a permutação escolhida

Quando uma coloração própria pode ser obtida através de um algoritmo guloso?

Uma coloração própria é uma coloração válida e, agora que ganhamos um pouco mais de conhecimento sobre colorações gulosas, vamos à pergunta central deste trabalho. Seja $G = (V, E)$ um grafo $k-$colorível e c uma $k-$coloração própria de G, a coloração c é uma coloração gulosa? Note que caso a resposta seja sim, a condição abaixo é satisfeita:

Para todo $v \in V(G)$, se $c(v) = i$ onde $0 < i \leq k$ então para todo j com $0 < j < i$ existe um vértice u vizinho de v com $c(u) = j$. Em outras palavras, se na coloração c um vértice v possui a cor i, então ele possui pelo menos um vizinho com cada uma das cores menores que i.

Sabendo disso observe a coloração exibida na Figura 1b, você acha que ela é uma coloração gulosa? Pense um pouco antes de continuar lendo. A resposta é não. E o motivo para isso está no vértice D, onde $c(D) = 3$ mas D não possui nenhum vizinho com a cor 2. Dessa forma independentemente da permutação escolhida a "maior"cor que D poderia assumir em uma coloração gulosa é 2.

Agora vamos olhar para a coloração gulosa da Figura 2a. Note que, se escolhêssemos a permutação AEFBDC, o resultado seria o mesmo. Porque isso acontece? Mais uma vez pense um pouco antes de continuar. Nesse caso, o que fizemos foi separar os vértices por *classes de cores*. Repare que os vértices $\{A, E, F\}$ foram aqueles coloridos com a cor 1, os vértices $\{B, D\}$ com a cor 2 e, por fim, o vértice C com a cor 3. A coloração de um grafo é gulosa se, para cada vértice v de cor c, houver vizinhos de todas as cores menores que c. E é essa a estratégia que iremos implementar.

Você deve usar esse problema e a estratégia de solução proposta para avaliar comparativamente algoritmos de ordenação. Mais ainda, você deve propor um método de ordenação (que pode ser adaptado a partir de outros métodos). Note que, para cada grafo e coloração a serem avaliados, o número de cores é conhecido e isso permite pensar na utilização de algoritmos mais eficientes que não demandam comparações.

O que você deve implementar

Entrada e saída

As instancias serão lidas pela pela entrada padrão. Consistem na escolha do método de ordenação a ser utilizado e na leitura dos dados do grafo e da coloração. Essa leitura será feita da seguinte forma:

- Primeira linha contendo um caractere c e um inteiro n, indicando respectivamente qual método de ordenação deve ser utilizado e a quantidade de vértices de G. Os valores que c pode receber são:
  - **b**: Método da bolha ou *bubble sort*.
  - **s**: Método da seleção ou *selection sort*.
  - **i**: Método da inserção ou *insertion sort*.
  - **q**: *quicksort*.
  - **m**: *mergesort*.
  - **p**: *heapsort*.
  - **y**: Seu próprio método de ordenação.
- As próximas n linhas possuem a vizinhança de cada vértice. Cada uma começa com um inteiro m indicando quantos vizinhos o vértice possui seguido de m inteiros indicando seus vizinhos.
- A última linha contém n inteiros indicando as cores de cada um dos vértices.

Caso a coloração não seja uma coloração gulosa sua saída deve ser 0. Caso contrário, imprima 1 seguido da permutação dos vértices utilizada. Em caso de empate na classe de cor, os vértices devem ser ordenados pelo rótulo.

Agora é a sua vez!

Além de implementar todos os métodos de ordenação listados na seção anterior, adaptados a sua estrutura de dados de escolha para armazenar os vértices do grafo, você deverá desenvolver e implementar seu próprio método de ordenação para este problema! Você deve definir bem sua estratégia e ela deve ser minimamente diferente dos outros métodos

implementados neste trabalho. Você pode se inspirar nos outros métodos vistos em sala ou tentar uma estratégia completamente nova. Seu algoritmo não precisa ser o mais eficiente de todos, mas deve ser capaz de executar em tempo polinomial no tamanho do grafo, e deve executar rápido o suficiente para ser aprovado pelos casos de teste propostos na VPL de entrega.

Comparação experimental!

Crie um conjunto de entradas amplo com grafos e conjuntos de cores de vários tamanhos. Execute testes com cada um dos métodos de ordenação (incluindo o que você criou/adaptou). Que informações você consegue extrair desses dados comparando os métodos? A verificação se a coloração é gulosa impacta nas suas comparações? Instrumente seu código separando as fases e refaça os experimentos. Argumente como as propriedades dos métodos como adaptabilidade e estabilidade interferem no resultado. Apresente gráficos que corroboram com seus argumentos. Como o seu método se compara com os métodos clássicos? Baseando-se em todos os resultados que obteve, qual dos métodos você acredita ser o mais aconselhável para resolver este problema?

Exemplos

Exemplo 1:

```
m 6
3 1 2 3
3 0 2 4
3 0 1 5
1 0
1 1
1 2
1 2 3 2 1 1
```

Como o caractere c vale "m", devemos utilizar o *mergesort* para fazer a ordenação dos vértices.

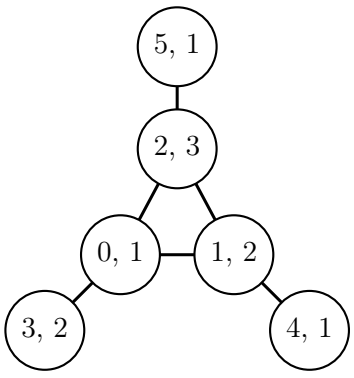


Figura 3: Representação visual do grafo do exemplo 1 (os valores escritos nos vértices simbolizam respectivamente o rótulo e a cor utilizada)

Saída esperada:

```
1 0 4 5 1 3 2
```

Exemplo 2:

```
q  6
3  1  2  3
3  0  2  4
3  0  1  5
1  0
1  1
1  2
1  2  3  3  1  1
```

Como o caractere c vale "q", devemos utilizar o *quicksort* para fazer a ordenação dos vértices.

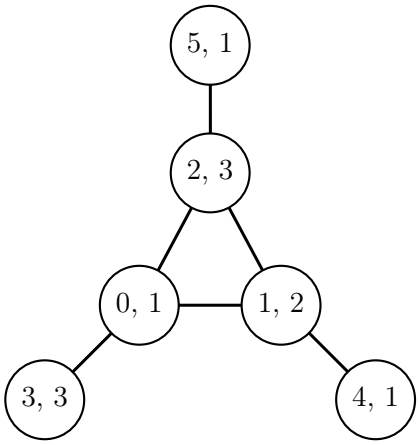


Figura 4: Representação visual do grafo do exemplo 2 (os valores escritos nos vértices simbolizam respectivamente o rótulo e a cor utilizada)

Saída esperada:

```
0
```

Exemplo 3:

```
y 5
2 1 4
2 0 2
2 1 3
2 2 4
2 0 3
1 2 1 3 2
```

Como o caractere c vale "y", devemos utilizar o seu próprio método para fazer a ordenação dos vértices.

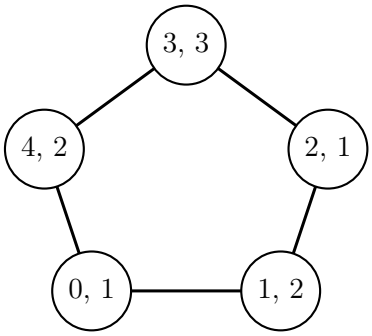


Figura 5: Representação visual do grafo do exemplo 3 (os valores escritos nos vértices simbolizam respectivamente o rótulo e a cor utilizada)

Saída esperada:

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

Exemplo 4:

```
p 5
2 1 4
2 0 2
2 1 3
2 2 4
2 0 3
1 2 1 5 2
```

Como o caractere c vale "p", devemos utilizar o *heapsort* para fazer a ordenação dos vértices.

Saída esperada:

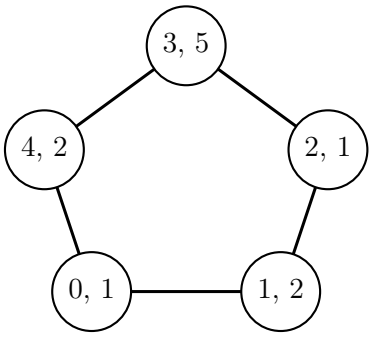


Figura 6: Representação visual do grafo do exemplo 4 (os valores escritos nos vértices simbolizam respectivamente o rótulo e a cor utilizada)

```
0
```

Documentação

A documentação do trabalho deve ser entregue em formato **PDF** e também **DEVE** seguir o modelo de relatório que será postado no Moodle. Além disso, a documentação deve conter **TODOS** os itens descritos a seguir **NA ORDEM** em que são apresentados:

1. **Capa:** Título, nome, e matrícula.
2. **Introdução:** Contém a apresentação do contexto, problema, e qual solução será empregada.
3. **Método:** Descrição da implementação, detalhando as estruturas de dados, tipos abstratos de dados (ou classes) e funções (ou métodos) implementados.
4. **Análise de Complexidade:** Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.
5. **Estratégias de Robustez:** Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
6. **Análise Experimental:** Apresenta os experimentos realizados em termos de desempenho computacional e localidade de referência, assim como as análises dos resultados.
7. **Conclusões:** A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.
8. **Bibliografia:** Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.
9. Número máximo de páginas incluindo a capa: 10

Não se esqueça de descrever de forma concisa como o método que você criou. Tome cuidado quando for demonstrar sua complexidade em número de comparações e de movimentações. Tente encontrar os limitantes mais estreitos o possível para a complexidade. Não se esqueça de argumentar sobre a adaptabilidade e estabilidade do método, justificando suas afirmações.
A documentação deve conter a descrição do seu trabalho em termos funcionais, dando foco nos algoritmos, estruturas de dados e decisões de implementação importantes durante o desenvolvimento.

Evite a descrição literal do código-fonte na documentação do trabalho.

Dica: Sua documentação deve ser clara o suficiente para que uma pessoa (da área de Computação ou não) consiga ler, entender o problema tratado e como foi feita a solução.

Como será feita a entrega

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é terminantemente vetado. Você DEVE utilizar a estrutura de projeto abaixo junto ao Makefile:

```
– TP
    |– src
    |– bin
    |– obj
    |– include
    Makefile
```

A pasta **TP** é a raiz do projeto; a pasta **bin** deve estar vazia; **src** deve armazenar arquivos de código (*.c, *.cpp, ou *.cc); a pasta **include**, os cabeçalhos (headers) do projeto, com extensão *.h, por fim a pasta **obj** deve estar vazia. O Makefile deve estar na raiz do projeto. A execução do Makefile deve gerar os códigos objeto *.o no diretório **obj** e o executável do TP no diretório **bin**. O arquivo executável **DEVE** se chamar **tp2.out** e deve estar localizado na pasta **bin**. O código será compilado com o comando:

```
make all
```

O seu código será avaliado através de uma **VPL** que será disponibilizada no moodle. Você também terá à disposição uma VPL de testes para verificar se a formatação da sua saída está de acordo com a requisitada. A VPL de testes não vale pontos e não conta como trabalho entregue. Um pdf com instruções de como enviar seu trabalho para que ele seja compilado corretamente estará disponível no moodle.
A documentação será entregue em uma atividade separada designada para tal no Moodle. A entrega deve ser feita em um único arquivo com extensão .pdf, com nomenclatura nome_sobrenome_matricula.pdf, onde nome, sobrenome e matrícula devem ser substituídos por suas informações pessoais.

Avaliação

O trabalho será avaliado de acordo com:

- Definição e implementação das estruturas de dados e funções - (30% da nota total)
- Corretude na execução dos casos de teste - (20% da nota total)
- Apresentação da análise de complexidade das implementações - (15% da nota total)
- Estrutura e conteúdo exigidos para a documentação - (20% da nota total)
- Indentação, comentários do código fonte e uso de boas práticas - (10% da nota total)
- Cumprimento total da especificação - (5% da nota total)

Se o programa submetido não compilar[1], seu trabalho não será avaliado e sua nota será 0. Trabalhos entregues com atrasos sofrerão penalização de 2^{d-1} pontos, com d = dias de atraso.

[1]Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.

Considerações finais

1. Comece a fazer esse trabalho prático o quanto antes, enquanto o prazo de entrega está tão distante quanto jamais estará.
2. Leia atentamente o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.
3. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
4. Plágio é crime. Trabalhos onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas (em relação a todos os envolvidos). Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na seção de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos.**

FAQ (Frequently asked Questions)

1. Posso utilizar alguma estrutura de dados do tipo Queue, Stack, Vector, List, e etc..., do C++? NÃO
2. Posso utilizar smart pointers? NÃO.
3. Posso utilizar o tipo String? SIM.
4. Posso utilizar o tipo String para simular minhas estruturas de dados? NÃO
5. Posso utilizar alguma biblioteca para tratar exceções? SIM.
6. Posso utilizar alguma biblioteca para gerenciar memória? SIM.
7. As análises e apresentação dos resultados são importantes na documentação? SIM.
8. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? NÃO
9. Posso fazer o trabalho em dupla ou em grupo? NÃO
10. Posso trocar informações com os colegas sobre a teoria? SIM.
11. Posso fazer o trabalho no Windows, Linux, ou MacOS? SIM.
12. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? SIM.