Resolução do problema EP02

Geraldo Rodrigues de Melo Neto
Gustavo Duarte Ventino
Maria Luisa Gabriel Domingues
Pedro de Araújo Ribeiro
Lucas Marques Pinho Tiago

O Problema:

A partir de um grafo conectado com as seguintes restrições:

- Um vértice não se conecta consigo mesmo;
- O grafo é não direcionado;
- O grafo é fortemente conectado;

Devemos verificar se tal grafo é bipartido, ou seja, “bicolorável”.

Entradas:

A entrada consiste em, nessa ordem:

- Número de vértices (1 < n < 200).
- Número de arestas.
- Pares de vértices que representa uma das arestas escritas.
- (Repete esse processo para quantos grafos quiser, e para parar de escrever grafos, digita-se 0 no campo de número de vértices).

Saídas:

Para cada grafo escrito no input, o programa deve responder com “BICOLORABLE.” ou “NOT BICOLORABLE.“, uma resposta por linha de output.

Nossa Resolução

Inicialmente, definimos dois TADs com classes : um para os grafos e outro para os vértices desses grafos.

- Graph.hpp : contém, de atributos, um array com os vértices do grafo, além do número de vértices e arestas.
    - Quanto aos métodos, temos Getters e Setters básicos para cada atributo, além de métodos de busca (tanto em largura quanto profundidade) que auxiliaram na conclusão de que o grafo é ou não bipartido, e de reset de marcações (de cor e visita) para efetuar ambos métodos de busca.

Nossa Resolução:

- Vertex.hpp: contém, de atributos, um array de vértices adjacentes, além da cor que carrega, sua marcação de visita ao vértice e sua identificação (dado no input).
  - De métodos, temos Getters e Setters básicos para cada atributo, além de métodos como addToAdjacency(), que inserem vértices no array de adjacência, gerando novas arestas.

```cpp
#include <vector>
class Vertex
{
private:
  int id;
  std::vector<Vertex*> adjacency;
  bool mark;
  int colour;

public:
  //Construtores:
  Vertex();
  Vertex(int id);
  //Getters:
  int getId();
  int getValue();
  bool getMark();
  std::vector<Vertex*> getAdjacency();
  int getColour(); // 0 = cinza 1 = preto, 2 = vermelho
  //Setters:
  void setId(int id);
  void setValue(int value);
  void setMark(bool mark);
  void setColour(int colour);
  //Adiciona o vertice v na lista de adjacencia
  void addToAdjacency(Vertex *v);
  //Printa as informacoes de id e valor do vertice atual
  void print();
  //Printa a lista de adjacencia do vertice atual
  void printAdjacency();
};
```

```cpp
#include "Vertex.hpp"

class Graph
{
private:
  std::vector<Vertex*> vertices;
  int size;
  int edges;
public:
//Construtores:
  Graph();
  Graph(int size, int edges);

//Faz toda a leitura de entrada e cria o grafo:
  static Graph* readGraph(int n, int m);

//Adiciona novo vertice na lista de vertices:
  void addVertex(Vertex* v);

//Dado um id retorna o vertice naquela posição:
  Vertex* getVertex(int id);

//Imprime todo o grafo:
  void print();

//Faz uma busca por profundidade e retorna caso o grafo é bipartido (true) ou não (false)
  bool dfs(int v);

//Faz uma busca por largura e retorna caso o grafo é bipartido (true) ou não (false)
  bool bfs(int v);

  void resetMark();
};
```

Método para resolução do problema:

Descreveremos, aqui, de forma mais detalhada, os métodos envolvidos na geração do output correto para o problema.

```
26  //Faz uma busca por profundidade e retorna caso o grafo é bipartido (true) ou não (false)
27    bool dfs(int v);
28
29  //Faz uma busca por largura e retorna caso o grafo é bipartido (true) ou não (false)
30    bool bfs(int v);
```

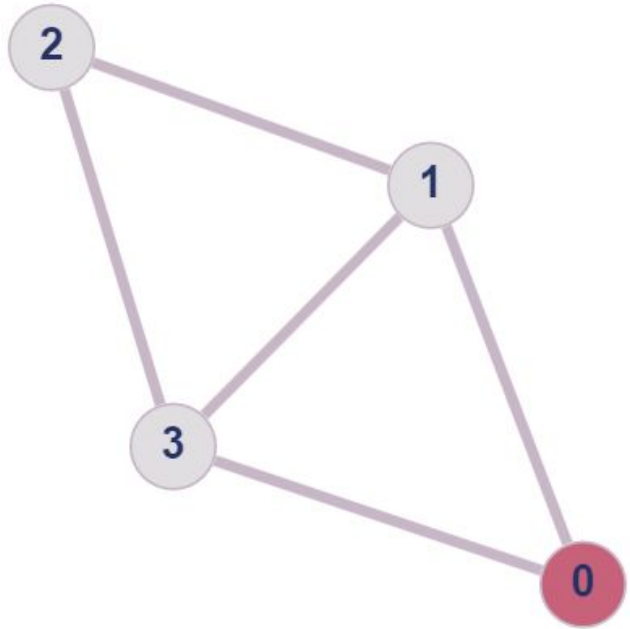
Busca em Profundidade (dfs):

Nossa busca em profundidade foi feita usando Pilhas para navegar no grafo, e eram feitas duas verificações:

1) Se o vértice adjacente não foi visitado ainda :
    a) Nesse caso, marcamos que o visitamos e adicionamos cor inversa ao do vértice em análise.
2) Se o vértice adjacente já foi visitado :
    a) Isso ocorre quando um vértice é adjacente a mais de um vértice. Nesse caso, comparamos as cores do vértice adjacente e do vértice em análise, e caso coincidissem, já retornava false (NOT BICOLORABLE) à main.
    b) Caso nenhum dos vértices caísse no caso 2, o método dfs retornava true (BICOLORABLE) à main.

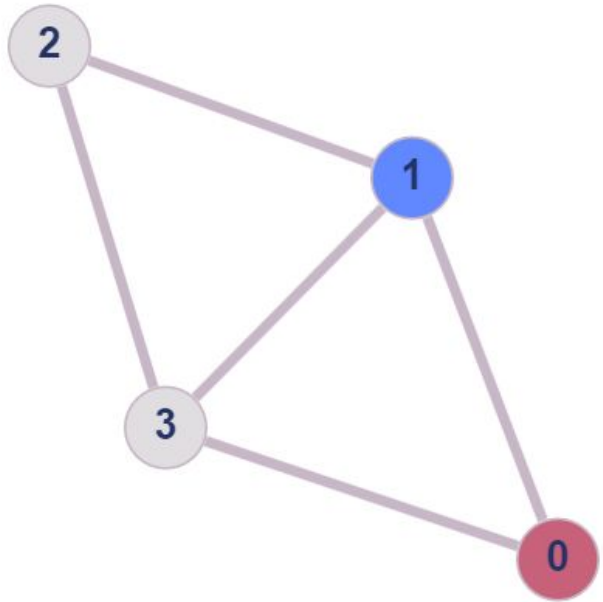
Busca em Profundidade (dfs):

```
62 bool Graph::dfs(int e){
63     int u;
64     std::stack<int> stack;
65     stack.push(e);
66     getVertex(e)->setColour(1);
```



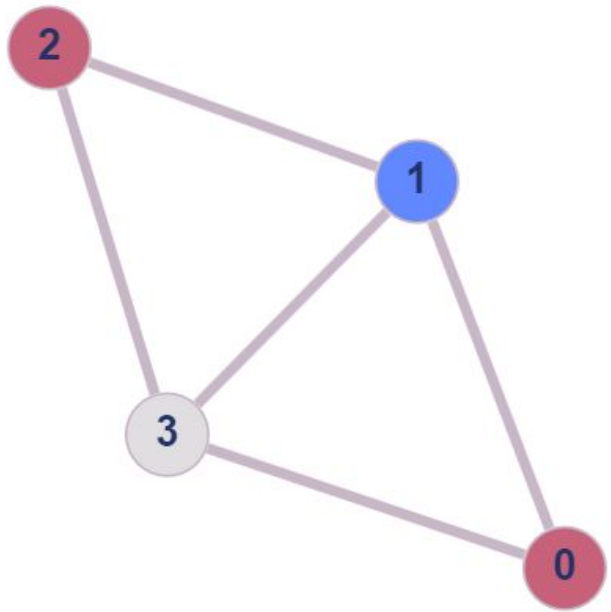
Busca em Profundidade (dfs):

```
67  while(!stack.empty()){
68    u = stack.top();
69    stack.pop();
70    Vertex *v = getVertex(u);
71    bool vmark = v->getMark();
72    int vcolour = v->getColour();
73    if(!vmark){
74      v->setMark(true);
75      for(auto i:v->getAdjacency()){
76        if(!i->getMark()){
77          if(vcolour == 1)
78            i->setColour(2);
79          else
80            i->setColour(1);
81          stack.push(i->getId());
82        }
83        else if(vcolour == i->getColour()){
84         return false;
85        }
86      }
87    }
88  }
```



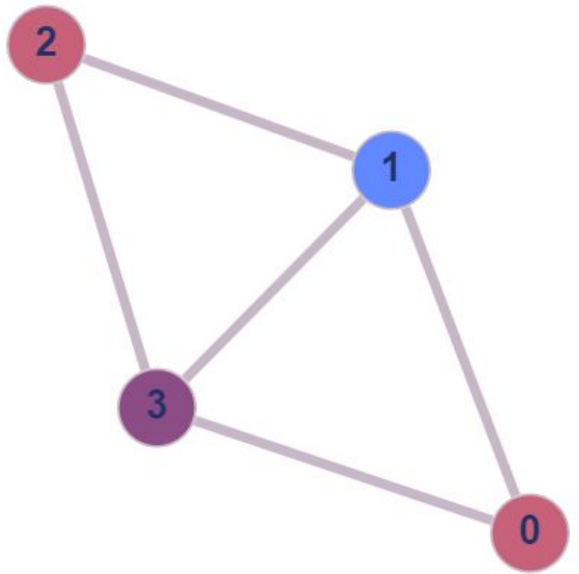
Busca em Profundidade (dfs):

```
67  while(!stack.empty()){
68    u = stack.top();
69    stack.pop();
70    Vertex *v = getVertex(u);
71    bool vmark = v->getMark();
72    int vcolour = v->getColour();
73    if(!vmark){
74      v->setMark(true);
75      for(auto i:v->getAdjacency()){
76        if(!i->getMark()){
77          if(vcolour == 1)
78            i->setColour(2);
79          else
80            i->setColour(1);
81          stack.push(i->getId());
82        }
83        else if(vcolour == i->getColour()){
84         return false;
85        }
86      }
87    }
88  }
```



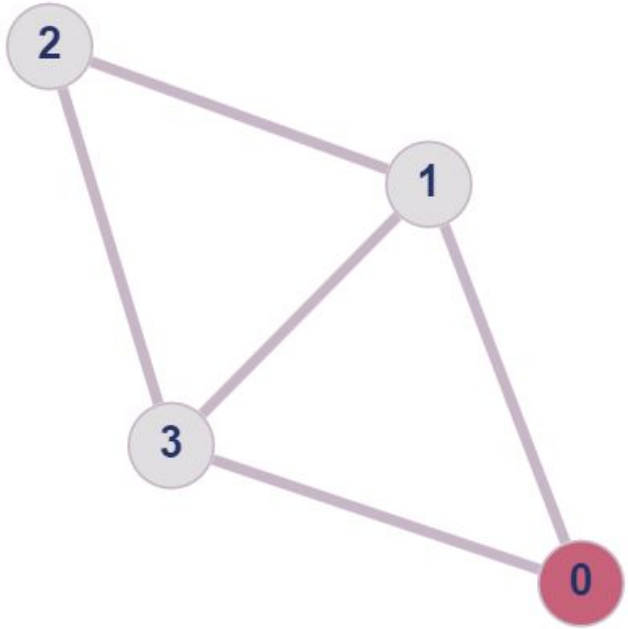
Busca em Profundidade (dfs):

```
67  while(!stack.empty()){
68    u = stack.top();
69    stack.pop();
70    Vertex *v = getVertex(u);
71    bool vmark = v->getMark();
72    int vcolour = v->getColour();
73    if(!vmark){
74      v->setMark(true);
75      for(auto i:v->getAdjacency()){
76        if(!i->getMark()){
77          if(vcolour == 1)
78            i->setColour(2);
79          else
80            i->setColour(1);
81          stack.push(i->getId());
82        }
83        else if(vcolour == i->getColour()){
84         return false;
85        }
86      }
87    }
88  }
```



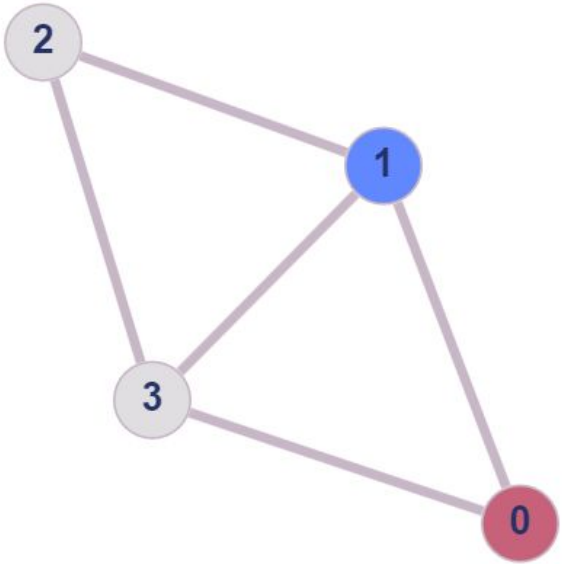
Busca em Largura (bfs):

```
93  bool Graph::bfs(int e)
94  {
95    int a;
96    bool flag = true;
97    std::queue<int> queue;
98    queue.push(e);
99    getVertex(e)->setColour(1);
100   getVertex(e)->setMark(true);
```



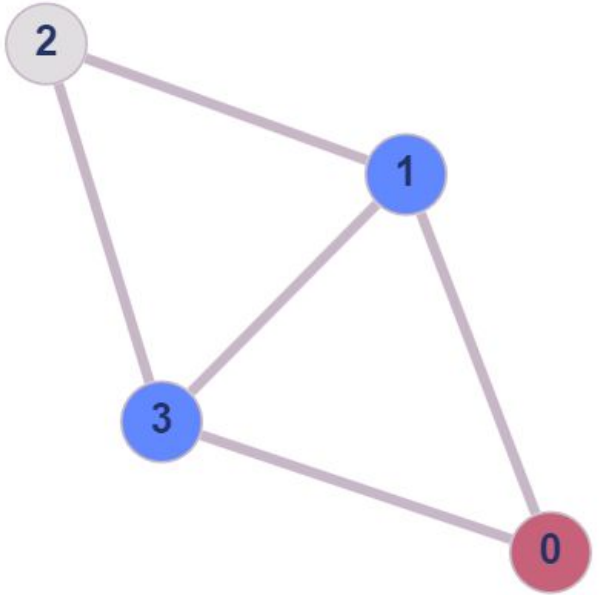
Busca em Largura (bfs):

```
101   while(!queue.empty())
102     {
103       a = queue.front();
104       queue.pop();
105       Vertex* v = getVertex(a);
106       bool vmark = v->getMark();
107       int colour = v->getColour();
108         for(auto i : v->getAdjacency())
109           {
110             if(!i->getMark())
111             {
112               i->setMark(true);
113               if(colour == 1)
114               {
115                 i->setColour(2);
116               }
117               else i->setColour(1);
118               queue.push(i->getId());
119             }
120             else if(colour == i->getColour())
121               return false;
122           }
```



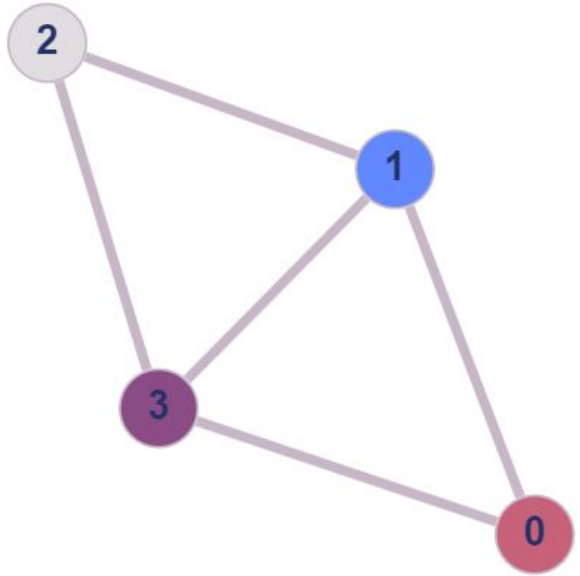
Busca em Largura (bfs):

```
101   while(!queue.empty())
102     {
103       a = queue.front();
104       queue.pop();
105       Vertex* v = getVertex(a);
106       bool vmark = v->getMark();
107       int colour = v->getColour();
108         for(auto i : v->getAdjacency())
109           {
110             if(!i->getMark())
111             {
112               i->setMark(true);
113               if(colour == 1)
114               {
115                 i->setColour(2);
116               }
117               else i->setColour(1);
118               queue.push(i->getId());
119             }
120             else if(colour == i->getColour())
121               return false;
122           }
```



Busca em Largura (bfs):

```
101   while(!queue.empty())
102     {
103       a = queue.front();
104       queue.pop();
105       Vertex* v = getVertex(a);
106       bool vmark = v->getMark();
107       int colour = v->getColour();
108         for(auto i : v->getAdjacency())
109           {
110             if(!i->getMark())
111             {
112               i->setMark(true);
113               if(colour == 1)
114               {
115                 i->setColour(2);
116               }
117               else i->setColour(1);
118               queue.push(i->getId());
119             }
120             else if(colour == i->getColour())
121               return false;
122           }
```



grafo criado do GraphOnline

FIM

Referências:

Imagens de grafos tiradas de: https://graphonline.ru/en/#

Repositório com os códigos fonte: https://replit.com/@Gezero/TG-EP02?v=1