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:

Dada uma quantidade **n** de voos entre cidades é necessário verificar se a partir de uma cidade específica é possível traçar um caminho onde o “Man of Mystery” continuará em movimento indefinidamente. Se sim, essa cidade é segura, se não ele está preso.

Entradas:

A entrada consiste em, nessa ordem:

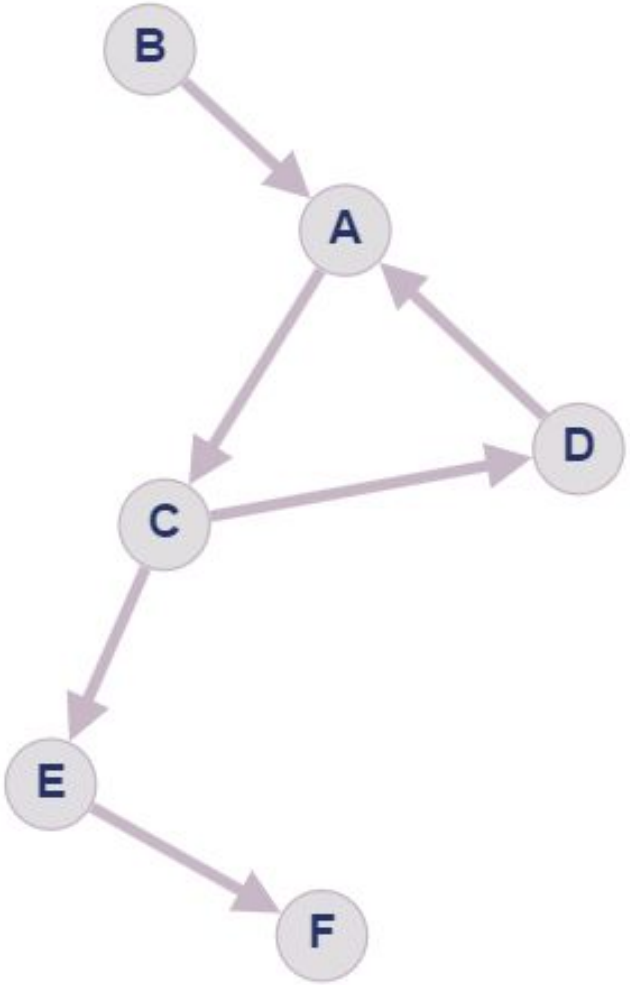
- Quantidade n de voos entre cidades;
- n linhas com a primeira cidade seguida de um espaço seguida da segunda cidade no modelo “**cidade_a cidade_b**”;
- as cidades que deseja verificar se são seguras.

Saídas:

O programa deve imprimir o nome da cidade e logo depois imprimir:

- “safe” se ela é segura;
- “trapped” se ela está presa.

Exemplo:



Nossa Resolução:

Decidimos resolver esse problema com Grafos pois podemos representar a existência de um caminho de A para B como as adjacências de um grafo direcionado.

Definimos dois TADs nos quais estão, respectivamente, classes presentes: uma para o grafo e outra para os vértices do grafo.

Nossa Resolução:

- Graph.hpp : contém, de atributos, um array com os vértices do grafo, além do número de vértices e arestas.
  - De métodos, contém Getters e Setters básicos para cada atributo, dois métodos para a geração do grafo e dois métodos referentes à verificação da existência de loops no grafo.
- Vertex.hpp: contém, de atributos, um array de vértices adjacentes, o valor do vértice (dado no input), seu id e se o vértice é seguro.
  - De métodos, contém Getters e Setters básicos para cada atributo, além de métodos como addToAdjacency(), que inserem vértices no array de adjacência, gerando novas arestas; há também o método de verificar se algum vértice da lista é seguro.

```cpp
#include <vector>
#include <string>
class Vertex
{
private:
  int id;
  std::vector<Vertex*> adjacency;
  std::string city;
  bool safe;

public:
  //Construtores:
  Vertex();
  Vertex(int id);
  Vertex(int id,std::string city);
  //Getters:
  int getId();
  int getValue();
  bool getSafe();
  std::string getCity();
  std::vector<Vertex*> getAdjacency();
  //Setters:
  void setId(int id);
  void setValue(int value);
  void setSafe(bool safe);
  void setCity(std::string city);
  //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();
  //Verifica se vizinho é seguro
  bool verificaViz();
};
```

```cpp
#include "Vertex.hpp"
#include <fstream>

class Graph
{
private:
  std::vector<Vertex*> vertices;
  int size;
  int edges;
public:
//Construtores:
  Graph();
  Graph(int size);
  Graph(int size, int edges);
//Getters:
  int getSize();
  int getEdge();
//Dado um id retorna o vertice naquela posição:
  Vertex* getVertex(int id);
//Setters:
  void setEdge(int edge);
  void setSize(int size);
//Faz toda a leitura de entrada e cria o grafo:
  static Graph* readGraph(int n, std::ifstream* file);
//Adiciona novo vertice na lista de vertices:
  void addVertex(Vertex* v);
//Imprime todo o grafo:
  void print();
//Verifica se a string (cidade) pertence à algum vertice do grafo
  bool verifyCity(std::string city, int *vertex);
//Operações que verificam se o vertice do grafo é seguro (vertice está em ciclo) ou não
  void verificaSafe();
  void buscaLoop(int id,bool vetAt[], bool vet[]);
};
```

Método para resolução do problema:

Optamos por dividir o problema em duas etapas, a leitura do grafo, que implica em verificar se uma cidade já foi registrada e atribuir as adjacências e a verificação de existência de loops no grafo e quais vértices fazem parte deles ou apontam para eles.

Leitura dos Dados:

```
34  //Faz toda a leitura de entrada e cria o grafo:
35  Graph* Graph::readGraph(int n, std::ifstream* file){
36    int vertex1 = 0, vertex2 = 0;
37    Vertex *v;
38    Graph *g;
39    std::string city1, city2;
40    g = new Graph();
41    int size = 0;
42    for(int i = 0; i < n; i++){
43      *file >> city1;
44      *file >> city2;
45      //std::cin >> city1;
46      //std::cin >> city2;
47      if(!g->verifyCity(city1, &vertex1)){
48        v = new Vertex(size, city1);
49        g->addVertex(v);
50        vertex1 = size;
51        size++;
52      }
53      if(!g->verifyCity(city2, &vertex2)){
54        v = new Vertex(size, city2);
55        g->addVertex(v);
56        vertex2 = size;
57        size++;
58      }
59      g->getVertex(vertex1)->addToAdjacency(g->getVertex(vertex2));
60    }
61    g->setSize(size);
62    return g;
63  }
```

```
65  //Adiciona novo vertice na lista de vertices:
66  void Graph::addVertex(Vertex* v)
67  {
68    vertices.push_back(v);
69  }
```

```
85  //Verifica se a string (cidade) pertence à algum vertice do grafo
86  bool Graph::verifyCity(std::string city, int *vertex){
87    for(auto i : vertices){
88      if(i->getCity().compare(city) == 0){
89       *vertex = i->getId();
90       return true;
91      }
92    }
93    return false;
94  }
```

Método para resolução do problema:

```
97  void Graph::verificaSafe(){
98    bool vis[size];
99    bool visAtual[size];
100   for(int i = 0; i<size;i++){
101     vis[i] = false;
102     visAtual[i] = false;
103   }
104   for(int i = 0; i<size;i++){
105     if(!vis[i])
106       buscaLoop(i,visAtual, vis);
107   }
108 }
109
```

```
47  bool Vertex::verificaViz(){
48    bool flag = false;
49    for (auto i : adjacency)
50      if(i->getSafe())
51        flag=true;
52    return flag;
53  }
```

```
110 void Graph::buscaLoop(int id,bool vetAt[], bool vet[]){
111   vetAt[id] = true;
112   for(auto i : vertices[id]->getAdjacency()){
113     if(i->getAdjacency().size()==0){
114     i->setSafe(false);
115     //std::cout << "fim: " << i->getId() << std::endl;
116     }
117     else if(vetAt[i->getId()]){
118     i->setSafe(true);
119     //std::cout << "bateu: " << i->getId() << std::endl;
120     }
121     else{
122       buscaLoop(i->getId(),vetAt, vet);
123       //std::cout << "flag: " << i->getId() << std::endl;
124       i->setSafe(i->verificaViz());
125     }
126   }
127   vertices[id]->setSafe(vertices[id]->verificaViz());
128   vet[id]=true;
129   vetAt[id] = false;
130 }
131
```

FIM

Referências:

Repositório com os códigos fonte: https://replit.com/@Gezero/TG2-EP02

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