Resolução do Problema EP01

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

O Problema:

Um colecionador de livros se encarrega de descobrir o alfabeto de um livro misterioso que encontrou, e precisa de ajuda computacional para descobrir a sequência de letras de um grupo de palavras já numa ordenação por “peso” de cada letra da linguagem usada no livro.

Entradas

A entrada consiste em escrever várias sequências de letras, dispostas uma em cada linha de input. Escreve-se um # para simbolizar o fim do grupo de sequências.

Saídas

O programa deve escrever a sequência de letras que representa o “alfabeto” da sequência de letras escrito no input.

Nossa resolução

Decidimos usar Grafos direcionados na resolução desse problema, tendo em vista que, analisando grafos desse tipo por uma ordem de precedência (ordenação topológica), facilmente podemos descrever uma ordem na hierarquia das letras dadas por input.

Definimos dois TADs nos quais duas classes estão 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.
  - De métodos, contém Getters e Setters básicos para cada atributo, além do método de ordenação topológica e busca em profundidade, que auxiliaram na conclusão sobre a hierarquia total das letras guardadas no grafo.
- Vertex.hpp: contém, de atributos, um array de vértices adjacentes, um número inteiro (id) para facilitar a localização do vértice em arrays tanto do grafo quanto das adjacências, e a letra que aquele vértice representa.
  - 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.

```cpp
#include <vector>

class Vertex
{
private:
  int id;
  char value;
  std::vector<Vertex*> adjacency;
public:
  //Construtores:
  Vertex();
  Vertex(char value);
  Vertex(int id,char value);
  //Getters:
  int getId();
  char getValue();
  std::vector<Vertex*> getAdjacency();
  //retorna o primeiro adjacente não colorido do vertice
  Vertex* getAdjacencyNotColored();
  //Setters:
  void setId(int id);
  void setValue(char value);
  //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();
};
```

```
8  class Graph
9  {
10 private:
11   std::vector<Vertex*> vertices;
12   int size;
13 public:
14   //Construtores:
15   Graph();
16   Graph(int size);
17   //Getters e Setters
18   int getSize();
19   void setSize(int size);
20   //Faz toda a leitura de entrada e cria o grafo:
21   static Graph* readGraph(std::ifstream* file);
22   //Adiciona novo vertice na lista de vertices:
23   void addVertex(Vertex* v);
24   //Dado um id retorna o vertice naquela posição:
25   Vertex* getVertex(int id);
26   Vertex* getVertexValue(char valor);
27   //Imprime todo o grafo:
28   void print();
29   void printVertices();
30   //procura no grafo se um vértice com o valor capturado já existe, e se
existe, guarda a posição dele no array de vertices do grafo em vertex
31   bool verifyChar(char Char, int *vertex);
32   //busca em largura, que define o output, guardando-o numa pilha
33   void dfs(int pos, bool *marcado, std::stack <Vertex*> *s);
34   //ordenação topológica, retorna a pilha feita no dfs para ser escrita na main
35   std::stack <Vertex*> ordenaVertices();
```

Método para Resolução do Problema

Decidimos capturar as sequências de letras e compará-las a pares, para descobrir relações de hierarquia entre algumas das letras dessas sequências.

Essas relações são guardadas no grafo, que depois de completo, é navegado via busca em profundidade, dentro do método de ordenação topológica, marcando a visita em cada vértice através de um vetor de valores Booleanos, cada posição representando um vértice do grafo.

Para cada vértice visitado, assim que todos os seus adjacentes também tenham sido visitados, ele é inserido numa pilha. Essa pilha, no fim de toda a busca, é retornada à main para ser escrita como output.

Método de resolução do problema:

Para inferir as relações de hierarquia entre letras de cada par de sequências dadas via input, usamos do mesmo conceito que nos permite dispor palavras em ordem alfabética: Ao comparar verticalmente duas palavras, o primeiro par de letras diferentes define qual das palavras é “maior”, devido à sua letra no par ser “maior” que a outra.

```
39  // Faz toda a leitura de entrada e cria o grafo:
40  Graph *Graph::readGraph(std::ifstream* file){
```

Método de resolução do problema:

A Ordenação Topológica, método que usamos para resolver o problema, consiste em ordenar linearmente todos os vértices de um grafo, de forma que cada vértice venha antes de seus descendentes. Esse método é ideal para esse problema justamente por ser capaz de ordenar todas as letras do conjunto de sequências baseando-se nas hierarquias estabelecidas, colocando uma letra “maior” antes das “menores” na sequência de output.

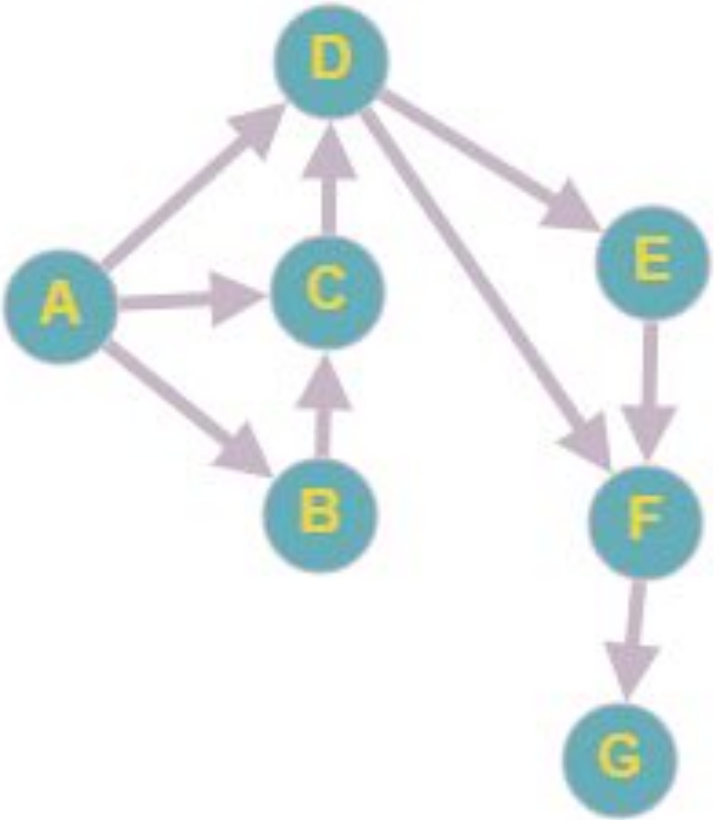
Como a Ordenação Topológica é feita?

Aplicamos esse método em conjunto com a busca em profundidade, usada para navegar nos vértices e, a cada grupo de adjacentes com todos pintados, inserir numa pilha passada de argumento o vértice “pai” desses adjacentes. Quando não há mais vértices que não foram visitados, o método de ordenação topológica retorna a pilha à main.

```
89  std::stack <Vertex*> Graph::ordenaVertices( ) {//ordenação topológica
90    std::stack <Vertex*> V;
91    bool marcado[vertices.size( )];
92    for(int i = 0; i < sizeof(marcado); i++) marcado[i] = false;
93    for(int i = 0; i < sizeof(marcado); i++)
94      {
95        if(!marcado[i]) dfs(i,marcado,&V);
96      }
97    return V;
98  }
```

```
78  void Graph::dfs(int pos, bool *marcado, std::stack <Vertex*> *s)
79  {
80    marcado[pos] = true;
81    for(auto j : vertices.at(pos)->getAdjacency())
82      {
83        //procura marcar cada vértice adjacente, identificando-o pelo ID
84        if(!marcado[j->getId( )]) dfs(j->getId( ),marcado,s);
85      }
86    s->push(vertices.at(pos));
87  }
```

Exemplo:



FIM

Referências:

Repositório com os códigos fonte: https://replit.com/@MariaLuisaGabri/S02EP01?v=1

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