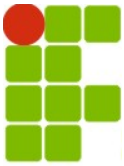

INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: GRAFOS – INTRODUÇÃO

Prática 10

Parte 0: Preparação

Passo 1: Crie um novo projeto chamado **Pratica10**.

Passo 2: Adicione os arquivos que acompanham a prática ao projeto.

> **graph.h** e **graph.cpp:** declaração e implementação do grafo, respectivamente.
>
> **main.cpp**: função *main()* que cria e testa o grafo.

Passo 3: Estudando as estruturas usadas.

> Nessas práticas são usadas as estruturas `vector`, `set`, `list`, `queue` da STL (*Standard Template Library*) de C++; veja a documentação delas se necessário, por exemplo:
>
> - https://en.cppreference.com/w/cpp/container/list
> - https://www.cplusplus.com/reference/list/list/

Parte 1: Implementando o Grafo

> **ATENÇÃO: Nas funções abaixo, os vértices passados como parâmetros devem ser verificados quanto a validade (valor entre 0 e nVert – 1); lançar exceção se inválidos.**

Passo 0: Estude a função `print()` do grafo.

> Essa função faz uso das funções seguintes que devem ser implementadas.

Passo 1: Implemente a função `edge()`.

> A função `edge(src, dst, w)` deve criar um objeto do tipo `Edge` com parâmetros `dst` (destino) e `w` (*weight*, peso) e adicionar à lista de vértices adjacentes associada ao vértice de origem (`src`).

Passo 2: Implemente a função `degree(vtx)`.

> A função `degree(vtx)` retorna o número de nós vizinhos do nó `vtx` (isto é, seu grau). Basta retornar o tamanho da lista associada ao nó (isto é, indexada por `vtx`).

Passo 3: Implemente a função `neighbors(vtx)`.

> A função `neighbors()` retorna uma lista (`list<int>`) contendo os nós vizinhos, identificados por números inteiros. Percorra a lista de arestas (`Edges`) associada com `vtx`, pegue o campo `dst` e adicione à lista para ser retornada.

Passo 4: Implemente a função `weight(src, dst)`.

A função `weight(src, dst)` retorna o peso da aresta entre `src` e `dst`. Percorra a lista de arestas associada ao vértice `src` até encontrar a aresta que leva ao vértice `dst`, e retorne o peso (`weight`).

Passo 5: Rode e teste a aplicação.

A função `print()` deve exibir a lista de vértices, com o grau do vértice entre parênteses, seguido da lista de nós vizinhos com o peso da aresta entre colchetes.

Parte 2: Testando se o grafo é conexo

Passo 1: Implemente a função `isConnected()` conforme descrito abaixo:

- A cada vértice, associe um identificador de grupo (número inteiro) ao qual o vértice pertence. Como o grupo é uma informação temporário, a forma mais simples é criar um array/vector de inteiros local ao método, indexado pelo número do vértice, que diz a qual grupo o vértice pertence. Inicialize o grupo de cada vértice como sendo o número do próprio vértice (isto é, inicialmente cada vértice pertence a um grupo que contém somente ele).
- Percorra todas as arestas do grafo, fazendo a junção dos grupos. Especificamente, percorra todos os vértices, e para cada vértice, varra todas as arestas. Para cada aresta (`edge`), faça com todos os vértices que pertencem ao grupo do destino (`edge.dst`) pertençam agora ao grupo da origem (`edge.src`) (isto é, modifique os valores no array de grupo).
- Ao final, para que o grafo seja conexo, todos os vértices devem pertencer ao mesmo grupo, do contrário o grafo não é conexo. Para verificar isso, veja se no array de grupos há mais de um valor diferente. (Dica: pegue o primeiro valor, e percorra o array até o fim; caso encontre um valor diferente, o grafo é não conexo; caso contrário, o grafo é conexo.).

Passo 2: (Desafio/Opcional). Implemente a função `isConnected()` de forma mais eficiente.

O procedimento descrito no Passo 1é uma versão simplificada do algoritmo *Union-Find* que roda em $O(N^2)$. A versão completa roda em $O(NlogN)$. Procure detalhes sobre o algoritmo e modifique a função `isConnected()`.

Parte 3: (Desafio/Opcional) Resolvendo o problema do Dominó

Passo 1: Crie e implemente a função `isEuler()`;

Para resolver o problema do dominó apresentado em sala é preciso testar se o grafo possui um ciclo de Euler. A 1ª condição para tal é que o grafo seja conexo (`isConnected()`); a 2ª é que no máximo dois vértices possuam grau ímpar. Crie um método `isEuler()` na classe `Graph` que diz se essas duas condições se aplicam.

Passo 2: Modifique a função `main()` e teste.

Adicione exemplos de grafos que contenham instâncias do problema do dominó (veja exemplo nos slides) e teste sua solução.