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 08

Parte 1: Implementando um Grafo Direcionado

Problema 1: Em **graph.cpp**, implemente as funções abaixo conforme a descrição:

Obs.: O código implementa um grafo direcionado simples.

- `edge(src, dst, w)`: adiciona arestas ao grafo. Adicione à lista de vértices adjacentes associada ao vértice de origem (`src`) um par com `{dst, w}`.
- `degree(vtx)`: retorna o número de nós vizinhos do vértice `vtx` (seu grau de saída, neste caso). Retorne o tamanho da lista associada ao vértice (indexada por `vtx`).
- `neighbors(vtx)`: retorna uma lista (`list<int>`) contendo os vértices vizinhos, identificados por números inteiros. Percorra a lista de arestas de adjacência de `vtx`, pegando o 1º campo como `dst` e adicione à lista para ser retornada.
- `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`).
- Teste rodando o **main.cpp**.

Problema 2: Em **graph.cpp,** 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, 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, no início cada vértice pertence a um grupo com somente ele).
- Percorra todos os vértices, e para cada vértice, varra todas as arestas, fazendo a junção dos grupos. Para cada aresta, faça com todos os vértices que pertencem ao grupo do destino pertençam agora ao grupo da origem (isto é, modifique os valores no array de grupo). Para essa função, a direção da aresta pode ser ignorada.
- 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).
- Teste rodando o **main.cpp**.

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

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

Parte 2: Resolvendo problemas utilizando grafos

Problema 1: Identifique roteadores sobrecarregados em uma rede (**network.cpp**).

> Dados dois grafos que representam a mesma topologia de rede, mas contendo respectivamente, a capacidade de transmissão entre dois roteadores, e o volume de tráfego atual de um para o outro, identifique quais roteadores estão com certeza sobrecarregados. Isto é, quais roteadores estão recebendo mais tráfego do que são capazes de transmitir. Assuma que o tráfego que entra num roteador deve sair dele por alguma saída (isto é, nenhum tráfego tem o roteador como destino).
>
> Implemente a função `overloaded()` em **network.cpp**, que retorna um `vector<int>` com os identificadores dos roteadores sobrecarregados.

Desafio (Opcional): Resolva o problema *Find the Town Judge* no LeetCode (LC997).

> Nesse problema, é dado um grafo direcionado na forma de matriz de adjacência que representa relações de confiança entre moradores de um pequena cidade. Nesta cidade há um (e apenas) "Juiz" oculto, que tem como característica ser confiado por todos e confiar em ninguém. Identifique o Juiz.

Problema 2: Implemente a função `is_euler()` em **domino.cpp**.

> Para resolver o problema do Dominó (OBI) é preciso testar se o grafo possui um ciclo (ou caminho) 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. Implemente a função `is_euler()` que diz se essas duas condições se aplicam.
>
> Rode o teste **domino.cpp** e veja se os resultados são os esperados.

Desafio (Opcional): Resolva o problema *Number of Provinces* no LeetCode (LC547).

> Nesse problema, é pedido para você retornar o número de províncias dado um conjunto de cidades. Nesse problema, uma província é definida como um conjunto de cidades são alcançáveis entre si (isto é, existe um caminho entre elas). É dado um grafo representado como matriz de adjacência dizendo que cidades estão diretamente conectadas umas as outras. Conte quantos conjuntos de cidades estão interconectadas entre si (isto é, quais os componentes conexos). **Dica:** modificação do algoritmo `isConnected()`.