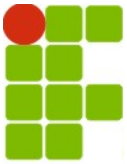

INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ALGORITMOS E ESTRUTURAS DE DADOS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: GRAFOS – BUSCAS E MENOR CAMINHO

Prática 11

OBS.: Esta prática é continuação da prática 10.
Pode ser feita no mesmo projeto ou em uma cópia.

Parte 1: Implementando Busca em Profundidade

Passo 1: Implemente a função privada `Graph::DFS()` em **graph.cpp**.

A função `dfs()` é a função pública chamada pelo usuário (na `main()`). Ela cria as estruturas auxiliares (vetor de nós visitados `visited[]`) necessárias uma vez e repassa para a função privada `DFS()`, que realiza a busca em profundidade de fato. Essa separação permite que a função `DFS()` se chame recursivamente, que é a implementação mais direta, sem ficar realocando as estruturas auxiliares a cada chamada. O *array* `visited[]` deve funcionar de forma que se um vértice `vtx` já foi visitado pela busca, `visited[vtx] == true`, do contrário `visited[vtx] == false`.

Siga o pseudocódigo do material de aula para implementar `DFS()`. Ela recebe, além de `visited[]`, uma referência para a lista `result` que deve conter ao final os nós na ordem em que foram atravessados na busca em profundidade. Isto é, a função **não** deve imprimir (exibir) os nós visitados mas adicionar a lista `result`.

Parte 2: Implementando Busca em Largura

Passo 1: Implemente a função privada `Graph::BFS()` em **graph.cpp**.

A busca em largura também foi quebrada em dois métodos: um público `bfs()` e um privado `BFS()`. Nesse caso a separação é apenas por organização e para manter a consistência com a busca em profundidade, mas não é estritamente necessária. Siga o pseudocódigo do material de aula para implementar `BFS()`. Essa função recebe o vetor `visited[]` e a lista (`result`) que deve conter ao final os nós na ordem em que foram atravessados na busca em largura. Use a fila (`queue<>`) da STL na sua implementação (ver *links* na prática 09).

Parte 3: Extraindo o Menor Caminho

Passo 1: Estude o código da função `Graph::spf()` em **graph.cpp**.

Essa função computa o menor caminho como descrito no material de aula. Ao final, o vetor `dist[]` contém a menor distância do nó `src` para o todos os outros nós (ex.: `dist[x]` deve ser a menor distância de `src` até o nó identificado por `x`), e o vetor `prev[]` contém o nó anterior no caminho de `src` até um dado nó (ex.: `prev[x]` deve retornar o nó que, no caminho de `src` até `x`, está imediatamente antes de `x`). Se `prev[x]` igual a `-1` indica que não há anterior; só é o caso para `prev[src]` ou se `x` não for alcançável a partir de `src` (não há caminho até `x`).

Passo 2: Implemente a função `Graph::path()` que extrai o menor caminho.

Essa função recebe como parâmetro uma lista vazia (`result`) a qual deve conter ao final a sequência de nós que compõem o caminho de `src` até `dst` (na ordem natural). Lembre-se que ao seguir o vetor de nós anteriores `prev[]` é obtido o caminho na ordem inversa; sua função deve retornar o caminho a ordem natural.