INSTITUTO FEDERAL
Catarinense
Campus Blumenau

CCA0925-Estruturas de Dados II

Bacharelado em Ciência da Computação

Prof. Dr. Paulo César Rodacki Gomes

Atividade prática - 01

1 Introdução

A presente atividade tem como objetivo iniciarmos a implementação de estruturas de dados de **grafos dirigidos e não-dirigidos**. À medida que formos avançando na disciplina, eventualmente podemos alterar ou implementar mais funcionalidades à essa primeira versão da estrutura.
Você pode implementar na linguagem de programação de sua livre escolha, embora o uso de uma linguagem orientada a objetos seja fortemente recomendável.
Os grafos são abstrações altamente poderosas e flexíveis para resolução de uma enorme gama de problemas. Assim, a implementação de estruturas de dados requer decisões de projeto compatíveis com o contexto de aplicação para o qual ela se destina. No momento não há necessidade de esgotarmos nossa análise sobre detalhes muito específicos de implementação porque nesta atividade estamos preocupados apenas em implementar uma estrutura básica. Futuramente podemos alterar nossa implementação para que seja mais adequada e/ou eficiente para determinados fins.
A atividade foi pensada assumindo que teremos abstrações para vértices, arestas e grafos. Possivelmente cada uma destas abstrações corresponderá a uma classe de objetos em sua implementação. Verifique também como você vai lidar com a questão de grafos dirigidos vs. não dirigidos. Eventualmente você pode decidir implementar uma classe base para grafos e especializá-la via herança para grafos dirigidos e não-dirigidos.

2 Descrição

Importante: a sua estrutura deve ser implementada como **listas de adjacência** ou **mapas de adjacência**, cabendo a você decidir qual prefere implementar.
O quadro mostrado na figura 1 mostra o conjunto mínimo de operações que você deve implementar neste exercício para **grafos simples não-dirigidos**. Métodos construtores e acessores não estão descritos no quadro, mas devem ser implementados. Você também deve desenvolver alguma forma de identificar vértices e arestas (por exemplo, por um identificador inteiro ou string). Note que algumas operações são gerais para qualquer tipo de grafos e outras são particulares a grafos dirigidos ou não-dirigidos (tais como operações para obter o grau de um vértice).
Além das operações descritas acima, você deve implementar uma operação para retornar uma string representando o grafo de forma textual.

Figura 1: Operações a serem implementadas

Operação	Descrição
getOrdem( )	retorna quantidade de vértices
getTamanho( )	retorna quantidade de arestas
vertices( )	retorna uma iteração de todos os vértices do grafo
arestas( )	retorna uma iteração de todas as arestas do grafo
insereV( )	instancia um novo vértice e o adiciona ao grafo
removeV(v)	remove o vértice v e todas as suas arestas incidentes
insereA(u, v)	instancia uma nova aresta incidente aos vértices u e v, e a adiciona ao grafo
removeA(e)	remove a aresta e
adj(v)	retorna uma iteração com todos os vértices adjacentes ao vértice u
getA(u, v)	retorna uma referência para a aresta de u para v ou null se os vértices não forem adjacentes. Para grafos não dirigidos, getA(u, v) e getA(v, u) produzem o mesmo resultado
grauE(v)	retorna o grau de entrada do vértice v em grafos dirigidos
grauS(v)	retorna o grau de saída do vértice v em grafos dirigidos
grau(v)	retorna o grau do vértice v em grafos não dirigidos. Alternativamente, pode-se usar os dois métodos anteriores em grafos não dirigidos de forma que grauE(v) e grauS(v) retornem o mesmo resultado
verticesA(e)	retorna o par de vértices que conectados à aresta e. Se o grafo for dirigido, o primeiro vértice do par é a origem e o segundo é o destino da aresta
oposto(v,e)	para um vértice v incidente à aresta e, retorna o outro vértice incidente à aresta
arestasE(v)	retorna uma iteração de todas as arestas de entrada do vértice v
arestasS(v)	retorna uma iteração de todas as arestas de saída do vértice v