Universidade Federal de Santa Catarina (UFSC)

Campus Reitor João David Ferreira Lima

Departamento de Informática e Estatística

Bacharelado em Ciência da Computação

Disciplina INE5413 - Grafos - 2023.2

Prof. Rafael Santiago

26/09/2023

Alunos:

João Pedro Resmer

Relatório - Atividade II - Grafos

1) Representação do Grafo

Além de uma abstração de vértice (Vertice) foram usadas duas classes, Graph e DiGraph, uma para representar grafos não dirigidos e uma para grafos dirigidos. Ambas usam as mesmas estruturas de dados:

1. Lista (V): usada para armazenar os objetos da classe Vertice, foi escolhida a estrutura list() do python por garantir a ordenação dos elementos;
2. Conjunto (vertices): usado para armazenar todos os índices de vértices existentes, escolhida, foi escolhida a estrutura de dados set() do python pois essa faz uso de hashing e acelera a operação de busca por elementos;
3. Lista (neighbors): usada para manter controle dos vizinhos de cada elemento, e possibilitar rápido acesso a essa informação;
4. Dicionário (edges): usado para armazenar as arestas e seu valor, escolhida por acelerar o acesso aos pesos das arestas por meio de hashing. Na classes Graph as chaves são do tipo fronzenset() do python o que permite a representação tanto de (u, v) quanto (v, u) em uma mesma posição, já na classe DiGraph são usadas tuplas (tuple()) para que haja diferenciação.

2) Componentes Fortemente Conexas

O algoritmo implementado foi o proposto em sala e usa apenas listas. Listas C e A foram usadas para controlar, repectivamente, os vértices já visitados e os antecessores de cada vértice nas buscas em profundidade. Além disso, foi utilizado um tipo list() como se fosse uma pilha para que na segunda DFS (Depth First Search) seja garantida a ordem correta. O tipo list() foi escolhido por possibilitar acesso aleatório eficiente e garantir a ordem dos elementos.

3) Ordenação Topológica

O algoritmo implementado foi o proposto em sala e é baseado em DFS. São utilizadas listas que armazenam: quais se um vértice já foi visitado, o antecessor de cada vértice, e os tempos de início e fim da visita de cada vértice. Novamente, o tipo list() foi escolhido por possibilitar acesso aleatório eficiente e garantir a ordem dos elementos.

4) Kruskal

O algoritmo de Kruskal foi implementado conforme proposto em aula. As estruturas utilizadas para isso foram:

1. Conjunto (A): representa a árvore formada;
2. Lista de conjuntos (S): usada para armazenar as subárvores construídas pelo algoritmo. O tipo list() foi escolhido por possibilitar acesso aleatório eficiente e o tipo set() (interno) foi escolhido por implementar as operações de união e diferença usadas no algoritmo.