Nome: Tális Breda

Relatório (Questão 4)

O trabalho inteiro foi feito usando Java e é orientado a objetos

Questão 1 - Componentes fortemente conexas

O algoritmo para encontrar componentes fortemente conexas é baseado no algoritmo de Cormen, apresentado nas anotações da disciplina. A estrutura de grafo usada é **GrafoDirigido**, que é uma simples adaptação da classe construída no trabalho anterior, porém sem a duplicação de arestas.
Para obter o resultado da DFS, é usada uma classe auxiliar **RespostaDFS**, com os atributos que devem ser retornados pela DFS.
Na DFS, as estruturas usadas foram listas para o conjunto de visitados, lista de tempos de entrada e de antecessores. Para o conjunto de tempos finais, foi usada uma **PriorityQueue** de **Tempo**, que é uma classe customizada criada para esse fim, tendo os atributos **vértice** e **tempo**. **PriorityQueue** funciona como um heap mínimo. Isso é usado para manter a ordem dos tempos finais, ao mesmo tempo que mantém a complexidade de tempo. Tendo o resultado da DFS, cria-se um Set com as arestas transpostas e um novo grafo com essas arestas, e inverte-se o conjunto de tempos finais, para que fiquem em ordem decrescente.
Após isso, a DFS é chamada novamente, mas para passar por essa nova lista. Um algoritmo simples é usado para identificar as subárvores dentro de A^T, e essas subárvores são impressas.

Questão 2 - Ordenação topológica

Aqui, as estruturas usadas são bastante simples. É usado o mesmo **GrafoDirigido**, e uma versão adaptada de **RespostaDFS**. Todos os conjuntos, exceto um, são representados por listas simples. O único que não é representado por listas simples é o conjunto dos vértices ordenados topologicamente, que é representado por uma **LinkedList**. A escolha da **LinkedList** foi por conta da capacidade de adicionar itens ao início da lista, facilitando o algoritmo.

Questão 3 - Kruskal

Para o algoritmo de Kruskal, foi utilizada a própria classe **Set** do Java, que funciona como um conjunto. Para a ordenação da lista de arestas, foi usado o método **stream().sorted()** da classe **List**, que tem complexidade de tempo O(n*log(n)) no pior caso.