Universidade Federal de Santa Catarina
Departamento de Informática e Estatística (INE)
Curso: Ciência da Computação
Disciplina: Grafos (INE5413)
Professor: Rafael de Santiago
Estudante: Guilherme Adenilson de Jesus (22100620)

Relatório A2

A implementação dos algoritmos requisitados foram feitos na linguagem de programação Python, devido ao maior conhecimento sobre sua *syntax* e estruturas. A seguir, é apresentado uma pequena justificativa das decisões feitas para uma das questões exigidas.

1. Representação

Os grafos desenvolvidos tiveram uma pequena variação em comparação com a atividade anterior. Dessa vez, por conta das Componentes Fortemente Conexas, a matriz de adjacência é uma "lista de listas", e os índices estão mapeados para o índice do vértice no arquivo lido mais uma unidade (index_matrix + 1 = index_arquivo). O motivo pela alteração é devido a necessidade de criação do grafo transposto, que seria muito custoso mantendo o padrão anterior. Ademais, como agora há dois tipos de grafos, cada um é uma especialização da classe Grafo, que possui todas as operações de representação.

2. Componente Fortemente Conexas

As estruturas que guardam vértices conhecidos, antecessores, ordem de finalização e tempos de início da visita são feitas com listas, para manter o padrão da matriz de adjacência. Contudo, a ordem de finalização funciona semelhante a uma pilha, para que não haja um maior custo de ordenação. Dessa forma, o último a concluir a visita sempre será o primeiro da pilha.

Para a formação do grafo transposto, é utilizado a biblioteca numpy, que possui um método de transposição de matriz. Para poupar um pouco de custo de cópia de dados, o grafo criado só tem a matriz de adjacência transposta e a quantidade de vértices. Como os demais atributos não são necessários para o algoritmo, eles mantiveram com os valores padrão de construção.

3. Ordenação Topológica

Para armazenar os vértices visitados e a ordenação topológica, foram utilizadas listas, permanecendo o padrão da matriz de adjacência. Devido à limitação do Python referente a argumentos de referência, foi necessário um custo a mais para a cópia dos dados enviados durante a recursão. Dessa forma, tanto a lista de vértices visitados e ordenação topológica precisam ser retornados no fim de cada execução do método *visita_ot()*, copiados, para manter o comportamento esperado.

4. Árvore Gerado Mínima (Kruskal)

Como objetivo de diminuir a quantidade de memória utilizada e otimizar o código, foi utilizado uma classe auxiliar CdElemento, sendo cada objeto dela uma referência a um vértice específico na lista S no início da execução. Essa classe substitui o uso de conjuntos, que normalmente são mais custosos de serem manipulados.

Para a ordenação das arestas do grafo pelo custo, utilizou-se o algoritmo QuickSort. Embora ele possua uma complexidade grande, $O(n^2)$, ele mantém um ótimo desempenho na maioria dos casos, além de ser fácil implementação.

Por fim, a escolha do algoritmo de Kruskal foi por ele ter sido visto primeiro que o de Prim, o que proporcionou mais tempo para ser revisado.