Relatório de desenvolvimento da A2

Aluno: Gabriel de Vargas Coelho

Matrícula: 20200400

Rodando os algoritmos

Para a execução correta dos algoritmos utilize no terminal os seguintes comandos:

1. Componentes Fortemente Conexas:

   python cfc.py [arquivo]

   exemplo: python3 cfc.py test.net
2. Ordenação Topológica:

   python ot.py [arquivo]

   exemplo: python3 ot.py test.net
3. Kruskal:

   python kruskal.py [arquivo]

   exemplo: python3 kruskal.py test.net

1. Representação

Na classe Grafo, são usadas 2 estruturas de dados, o vetor para os atributos de rótulo e de arestas existentes, e uma matriz |V|x|V| que armazena os custos entre as arestas. Essas estruturas foram escolhidas pois tornam a implementação dos demais algoritmos mais simples e de fácil compreensão.

2. Componentes Fortemente Conexas

O algoritmo utiliza 3 vetores para manter o registro dos vetores já conhecidos, o tempo final da visita, e o ancestral dos vértices, representados pela posição no vetor ancestral. Foram utilizados essas estruturas pela semelhança com os pseudo-códigos apresentados em aula.

Para a ordenação do vetor de tempo final de visita foi usado o algoritmo de merge sort e uma estrutura de dados baseada num dicionário python que armazena o tempo da visita, e o vértice em questão. O merge sort foi escolhido pela sua baixa complexidade computacional, e a estrutura utilizada foi escolhida como meio de manter um registro simples das informações úteis para o algoritmo cfc.

Para manter a conformidade com a saída apresentada no enunciado, também foi implementado uma função para a montagem do resultado apresentado no terminal. A função é recursiva e tem como objetivo montar vetores que armazenam somente vértices que participam de um mesmo cfc.

3. Ordenação Topológica

Para esse algoritmo foram utilizados: um vetor para registro de vértices conhecidos, e uma lista encadeada que armazena os vértices topologicamente ordenados. A lista encadeada foi escolhida pois possui complexidade computacional constante para a inserção de um elemento no início da lista.

4. Kruskal

A escolha pelo algoritmo de Kruskal se deu pela baixa complexidade de implementação deste, além de poder tirar proveito de estruturas já disponibilizadas pela linguagem python, como o set, que representa conjuntos. Sobre as estruturas de dados utilizadas, foram usados: um vetor que armazena os conjuntos que representam as árvores já montadas; e sets (conjuntos) tanto para o conjunto de arestas que fazem parte da árvore geradora mínima, quanto para os vértices que participam de uma sub-árvore. Também foi utilizado o algoritmo de merge sort nesse algoritmo, pelas mesmas razões já apresentadas.

5. Observação

O algoritmo de merge sort foi adaptado de: https://pt.wikipedia.org/wiki/Merge_sort#C%C3%B3digo_em_Python