UNIVERSIDADE FEDERAL DE SANTA CATARINA
Departamento de Informática e Estatística - INE
Ciências da Computação
INE5413 - Grafos

RELATÓRIO TÉCNICO

André William Régis (19200411)
Luiz Maurício do Valle Pereira (21104157)

Florianópolis, 29 de junho
2023

1 Introdução

Para visualizar o código fonte desse trabalho é necessário acessar github.com/Luiz276/grafos.

Para a execução do trabalho é necessário a instalação do `Poetry`. A seguir para a instalação das dependências é necessário executar na pasta do projeto:

```
$ poetry install
```

Finalmente para a execução do programa na raiz do projeto utilizasse:

```
$ poetry run python graph_solver OPTION [FILE...]
```

Ou para executar diretamente com o interpretador de python (dado que não há dependências indispensáveis):

```
$ python3 graph_solver/__main__.py OPTION FILE
```

Sendo `OPTION` substituído por um número e FILE por um caminho para o arquivo do grafo:

9. Resolve fluxo máximo
10. Resolve emparelhamento
11. Resolve coloração

`[FILE...]` admite nenhum, um ou vários arquivos de grafos, sendo os caminhos relativos à raiz do projeto. Caso nenhum arquivo seja especificado, o programa utilizará um grafo padrão "test/graph1.txt".

2 Estrutura de Dados

Vértices, arestas e o grafo como um todo foram representados por classes, como explicado a seguir:

- Vertex: Representa um vértice, contendo duas variáveis:
  - index - Armazena o index de um vértice
  - label - armazena o label de um vértice
- Edge: Representa uma aresta não orientada do grafo, armazenando vértices de origem e destino
- Graph: classe que representa um grafo, com uma lista de vértices V chamado de "vertices", uma lista de arestas E chamada de "edges", e uma lista "weights" que mapeia o peso de cada aresta.
  - Seus métodos representam operações esperadas de um grafo.
  - Listas foram escolhidas devido a representarem os dados de maneira mais simples e intuitiva, mesmo não sendo o caso ótimo para todas as operações realizadas.

Cada uma das estruturas de dados foram escolhidas por representarem de maneira intuitiva e eficiente os dados do programa, tornando o código o mais próximo possível dos pseudocódigos de exemplo passados em aula. Durante o desenvolvimento do código, testes

unitários foram utilizados para auxiliar no desenvolvimento de algumas partes. Testes se encontram no arquivo "`test/teste_ex1.py`".

3 Questões resolvidas

Todas as 3 questões foram resolvidas.

- Questão 1: Implementada no arquivo `t3_e1_edmonds_karp.py`, através da função `print_edmonds_karp`, baseia-se fortemente no exemplo fornecido na apostila da disciplina considerando `source` e `sink` como o primeiro e último vértice do arquivo do grafo, respectivamente. A saída segue o seguinte formato, com `s` sendo `source`, `t` sendo `sink` e `n` sendo o fluxo máximo:
  - `Fluxo máximo entre s e t: n`
- Questão 2: Resolvida no arquivo `t3_e2_hopcroft_karp.py`, através da função `print_hopcroft_karp`. A saída segue o seguinte formato, com `n` sendo o emparelhamento máximo:
  - `Emparelhamento máximo: n`
  - `Emparelhamentos: {lista contendo todos os emparelhamentos}`

  Vale notar que a lista apresenta 2 edges para cada emparelhamento, da seguinte maneira:
  - `Vertex(index=i, label='j'): Vertex(index=k, label='l')`
  - `Vertex(index=k, label='l'): Vertex(index=i, label='j')`

  O resultado mostrado ainda é correto, contendo apenas os emparelhamentos redundantes.
- Questão 3: Resolvida no arquivo `t3_e3_vertex_coloring.py`, através da função `print_vertex_coloring`, que utiliza um algoritmo greedy para realizar a coloração. A saída segue o seguinte formato, com `n` sendo o número de vértices no grafo, `i` sendo a cor utilizada para o vértice e `k` sendo o total de cores utilizadas:
  - `Vértice 1 -> cor 0`
  - `Vértice n -> cor i`
  - `Número cromático: k cores utilizadas`