INE5413: Atividade 2

Vitor Della Torre dos Santos

Abril, 2021

1 Introdução

A atividade 2 é a segunda atividade da disciplina INE5413, de Grafos. Usando a biblioteca de grafos criada no último trabalho, seu intuito era realizar a implementação de 3 algoritmos. Estes são:

1. O algoritmo de Componentes Fortemente Conexas;
2. O algoritmo de Ordenação Topológica; e
3. O algoritmo de Kruskal.

Os algoritmos enumerados serão descritos a partir da seção 3, visto que a seção 2 destina-se à descrição das recém realizadas alterações na classe `Graph`.

2 API: a classe `Graph`

Houve sérios problemas no uso de Kotlin durante a construção do segundo trabalho. Estes tiveram como causa uma recente atualização da interface de desenvolvimento IntelliJ e, consequentemente, a perda de diálogo entre o framework Maven e a IDE. Muito tempo foi empreendido numa tentativa de solucionar o problema, mas, priorizando o tempo de execução dos exercícios, optou-se por migrar o trabalho para uma outra linguagem: `Python`.

A API utilizada para o manuseio dos grafos foi feita pensando em ser o mais parecida possível com a implementada em Kotlin. Ainda faz-se uso de uma matriz de adjacências e, também, uma lista armazenadora de rótulos para cada um dos vértices de um dado grafo.

3 Componentes Fortemente Conexas

O algoritmo das componentes fortemente conexas foi aquele em que mais se empreendeu tempo. Houve erros em decorrência da ausência de passagem de referência na variável nomeada por `time`. Para que se pudesse contornar a situação, criou-se uma lista com uma única posição e, portanto, com um único valor; este é atualizado durante a execução do algoritmo e, enquanto mantido numa lista, realiza corretamente sua passagem por referência.[1]

Realiza-se uma Busca em Profundidade no grafo. Em seguida, cria-se um grafo transposto, a partir da inversão de todos os arcos do primeiro. Então, aplica-se uma outra Busca em Profundidade, desta vez usando o vetor referente ao "tempo de término de execução dos nodos" como *pivot*. Para aproximarmo-nos da solução do problema, nesta etapa, adicionou-se todos os valores do vetor a um dicionário, de modo que pudesse-se utilizar um valor da lista de término de execução, como chave, para encontrar o identificador numérico de um dado nodo, como valor. Organizou-se este dicionário em ordem decrescente a partir de suas chaves.

Por fim, retornam-se as componentes fortemente conexas para o grafo desejado.

```
[0, None, 4, 5, 6, 7, 0, None, 0, 0, None]
parzival@merlim:~/cco/grafos/a2$
```

Figure 1: Resultado do algoritmo de Componentes Fortemente Conexas. Valores marcados por `None` são pontos de partida das componentes e não possuem ancestrais.

4 Ordenação Topológica

O algoritmo de Ordenação Topológica teve tempo de implementação um tanto mais curto que o anterior. O algoritmo anterior fez uso da função responsável por executar a Busca em Profundidade, a qual possui uma função auxiliar denominada `_visit`. O algoritmo de Ordenação Topológica, contudo, fez uso somente desta última.

[1] Anteriormente, com a ausência desta lista, `time` sofria passagem por valor.

A diferença marcante de `_visit` neste algoritmo é o uso efetivo da pilha, passada por parâmetro, apenas inicializada, mas vazia. É ela a responsável por armazenar os valores e, por fim, é chamada na função `show` para apresentar cada um dos valores, da esquerda para à direita.

```
parzival@merlim:~/cco/grafos/a2$ python3 main.py
"Acordar" -> "Tomar café da manhã" -> "Escovar os dentes" -> "Ler o jornal" ->
"Tomar banho" -> "Vestir camisa" -> "Calçar meias" -> "Vestir roupa de baixo" -
> "Vestir calças" -> "Calçar tênis" -> "Sair" -> None
parzival@merlim:~/cco/grafos/a2$
```

Figure 2: Apresentação do resultado da Ordenação Topológica.

5 Kruskal: árvore geradora mínima

Infelizmente, não conseguiu-se implementar o algoritmo de Kruskal.

6 Conclusão

Por meio da elaboração do trabalho, foi possível adquirir base teórico-prática para futuras implementações de grafos e estruturas de dados mais complexas. O código-fonte pode ser encontrado no Moodle.