Universidade Federal de Santa Catarina - UFSC
Departamento de Informática e Estatística (INE)
Ciência da Computação
Disciplina INE5413 - Grafos

Caio Prá Silva (21203773)
Fillipi Mangrich Costa de Souza (21202110)
Leonardo Lima Appio (21101963)

Relatório Atividade 1

Para testar o código de cada questão basta descomentar o código correspondente disponível no arquivo main.py.

1. Representação

Para o problema de representação dos grafos não-dirigidos e ponderados, optou-se por criar classes para vértices e arestas. A classe Vertice possui um atributo de rótulo, índice, grau e uma lista de vizinhos, permitindo que os vizinhos do vértice sejam percorridos em algoritmos que precisam dessa funcionalidade. Aresta possui atributos “u” e “v”, que são vértices que a aresta conecta, e um peso. A classe Grafo possui um dicionário para armazenar os vértices, com a chave sendo o índice dele e o valor o objeto vértice; há um dicionário para armazenar as arestas, com os índices dos vértices em uma tupla servindo como chave, e o objeto como valor. Como é implementado, a estrutura de dicionário é como um hashmap, ou seja, possui complexidade de tempo em caso médio O(1) para as operações de consulta e inserção, com O(n) como pior caso, mas bastante improvável devida a natureza da função de hash usada, que raramente ocorrem colisões.

2. Busca

Armazenamos os vértices já visitados, as distâncias e os antecessores em dicionários do Python para facilitar o acesso aos elementos, visto que os itens em um dicionário no Python são acessados de [1..n], sendo n o tamanho do dicionário, assim não precisamos fazer -1 para acessar os itens. Utilizamos, também, uma fila a partir da classe Queue() embutida no Python, escolhemos ela pois como já estava na linguagem não precisamos criar uma estrutura de Fila. Para facilitar na ordenação do retorno no formato requisitado de saída utilizamos a estrutura defaultdict da biblioteca collections, escolhemos usar ela pelo fato dela não lançar uma exceção do tipo KeyError, diferente do dict padrão do Python.

3. Ciclo Euleriano

Primeiramente, importamos as bibliotecas necessárias, usamos a nossa classe Grafo implementada anteriormente. Em seguida, definimos a classe CicloEuleriano com um construtor que recebe um objeto Grafo. O método isEulerian() verifica se cada vértice do grafo tem grau par, retornando True se for verdadeiro e False caso contrário. O método hierholzer() é o principal método da classe que implementa o algoritmo de Hierholzer. Ele inicializa um dicionário arestas_visitadas com as arestas do grafo. Em seguida, cria uma lista arestas com as arestas do grafo e seleciona um vértice inicial v. O método buscarSubcicloEuleriano() é responsável por encontrar um sub ciclo euleriano a partir de

um vértice inicial v e as arestas visitadas arestas_visitadas. Ele usa um loop e verifica se todas as arestas foram visitadas, retornando False se ainda houverem arestas não visitadas. O método printEulerian() simplesmente imprime 0 se o grafo não for euleriano ou 1 e o ciclo euleriano encontrado caso contrário. Optamos por utilizar essas estruturas para facilitar a implementação utilizando como base o algoritmo disponibilizado na apostila.

4. Dijkstra

O algoritmo de Dijkstra foi modelado dentro de uma classe Dijkstra, nela contendo os métodos responsáveis pela execução do algoritmo e saída, conforme especificada no enunciado. Foram usadas novamente estruturas de dicionário para representar os distâncias (D), os vértices visitados (C) e os antecessores de um dado vértice (A), todos sendo atributos da classe, além do vértice de entrada e o grafo a qual pertence.

Na execução do algoritmo, o dicionário que armazena os vértices não visitados a cada iteração do laço que percorre os vértices não visitados permite armazenar de forma mais eficaz tais informações, podendo rapidamente encontrar o objeto do vértice a partir de seu índice, usado como chave para essa estrutura. A chave encontrada com o menor valor e para encontrar o vértice procurado, ou seja, o que possui tal chave, é feito percorrendo o dicionário e comparando os atributos desejados para cada um.

No método “__caminho”, chamado pelo método “__print_result”, após encontrados os caminhos e distâncias dos vértices no grafo G, a formação dos caminhos é estruturada em um dicionário, com chave o vértice final e valor o caminho, que por sua vez é uma lista, indo do valor inicial, passado como parâmetro da função, até um vértice final “k”, a qual foi optado por uma lista por haver necessidade de colocar os dados em uma ordem, inserindo sempre os antecessores no começo da lista, como uma fila, para então formar o caminho.

5. Floyd-Warshall

Neste algoritmo utilizamos dicionários pelo mesmo motivo de acesso explicado na questão 2. Utilizamos uma cópia da matriz de adjacências para não alterar os itens da matriz original. Na saída foi utilizado uma lista para facilitar a formatação dos campos retirados da matriz gerada com os caminhos mínimos.