Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Ciência da Computação

INE5413 – Grafos

Professor: Rafael de Santiago

Equipe:

Anthony Bernardo Kamers (19204700)

Julien Hervot de Mattos Vaz (19202276)

Ricardo Jensen Eyng (19203166)

Relatório - A1

Florianópolis, 25 de novembro de 2021

Questão 1: Foram utilizadas duas classes para abstrair a leitura e incorporação dos métodos do grafo, a Grafo e a Node. o atributo de Grafo “grafo” é uma lista de Node’s. Cada Node representa um vértice. Por comodidade (para separar quando precisar), em Node, decidimos separar os vértices vizinhos em listas (o edgesEntrada para vértices que entram no vértice atual e o edgesSaida para os que saem do vértice atual). Vale contar que isso foi feito para fazer a separação em grafos dirigidos. Cada um dos atributos, edgesEntrada e edgesSaida armazenam uma lista de 2 elementos, em que o primeiro elemento (índice 0) é o id do vértice e o segundo elemento (índice 1) é o valor do arco entre os vértices (para grafos dirigidos).

Questão 2: Para fazer os itens específicos da busca em largura, fizemos uma classe auxiliar NodeTree (para saber o nível do vértice e quem é seu pai). Além disso, iniciamos 3 filas (arvore, passados e fila). A fila é uma fila de NodeTree (a começar pelo vértice passado na função). Após isso, seguindo o algoritmo, enquanto a fila não está vazia, é pego o primeiro elemento da fila e armazenado na arvore e, para manter um histórico dos vértices que já passados, adicionamos seu id em passados.

Questão 3: Para testar o ciclo euleriano, precisamos fazer uma classe auxiliar Aresta (contendo os vértices de entrada/saída e se já foi passado por essa aresta na busca do ciclo (como atributo hasPassed)). Usando o Grafo, o percorremos e criamos uma lista de Aresta. Percorremos então essas arestas usando um vértice arbitrário (utilizamos o primeiro). Para busca de um subciclo euleriano, iniciamos uma lista, chamada ciclo. Nessa lista, na primeira parte da função, são adicionados os id’s dos vértices que compõem o ciclo. Na segunda parte (em que são testadas as arestas que não foram percorridas ainda), é adicionado uma lista (por recursividade da função buscar_subciclo_euleriano). Depois, usamos uma função que transforma toda lista em número, fazendo com que a variável ciclo fique apenas com inteiros (os id’s dos vértices que compõem o ciclo).

Questão 4: Para encontrar o caminho mínimo usando Bellman-Ford, criamos duas classes auxiliares (a Vertice, que tem o id do vértice, a distância até o vértice passado (começando por infinito) e seu antecessor (iniciado com valor nulo) e a classe Aresta, que tem dois vértices (Vertice) de entrada/saída e o valor/peso entre eles). Criamos então uma lista de vértices (Vertice) e depois, com esses, é criada uma lista de arestas (Aresta). Depois percorremos essa lista de arestas e, caso seja satisfeita a condição das distâncias entre as arestas, é aplicado o algoritmo de relaxamento.

Questão 5: Para encontrar o caminho mínimo de todos os vértices para todos os vértices de Bellman-Ford, é criada uma matriz (variável matriz0) de qtdVertices x qtdVertices. Nesta matriz, são inseridos os pesos entre os vértices (i,j). No caso, se for o próprio vértice (i=j) é 0, se há aresta entre os vértices, o peso dessa e, caso não haja aresta e não seja o próprio vértice, é adicionado o valor de infinito (para ajustar nos cálculos depois). Depois é percorrida essa matriz (com 3 loops) e feitos os ajustes, sempre pegando o menor caminho entre os vértices).