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 - A3

Florianópolis, 15 de março de 2022

Questão 1:
Foram criadas as estruturas conforme o algoritmo, uma lista de booleanos para manter os vértices visitados, uma lista de inteiros de índices dos antecessores, e uma fila para manter os vértices já visitados. Para finalizar e colocar na tela o resultado do fluxo máximo, é preciso colocar os pesos dos arcos em uma lista para, posteriormente, pegar o menor entre eles.

Questão 2:
Para manter o histórico de cada vértice que passou, criamos um dicionário para a distância (com chave o id do vértice e valor a distância do mesmo) e um para os "mates" (com chave o id do vértice e valor o seu respectivo "parceiro"). Foi preciso definir um X, para simular o conjunto bipartido de vértices e, como tal, foi utilizada uma lista. Na função BFS, foi criada uma fila para fazer o algoritmo de busca em largura. Na função DFS, foi utilizada uma lista de vizinhos de saída de cada vértice (*grafo.vizinhosSaida(x)*).

Informação questão 3:
Ao tentar realizar o exercício conforme o algoritmo indicado (coloração de Lawler), não conseguimos obter o resultado esperado. Como tal, tentamos realizar de outra maneira, utilizando o algoritmo *greedy coloring*. Também não conseguimos obter o resultado esperado, mas colocaremos as estruturas de dados utilizadas em cada um.

Questão 3:
A fim de utilizar uma biblioteca que gera o conjunto máximo independente, foi preciso recriar a classe Grafo, utilizando a biblioteca *networkx* (chamamos de Grafo1). Como primeira ação do algoritmo de Lawler, vamos criar uma lista de dicionários, chamada *table_mark* (contendo as chaves key, sendo uma lista de booleanos que compõem a iteração, a chave elements sendo uma lista dos elementos que compõem a iteração (de acordo com os booleanos), e uma chave value tendo como valor inicial infinito, aqui representado como *sys.float_info.max*). Após isso é iterado cada posição de *table_mark* e fazemos um subgrafo (lista de tuplas) para cada elemento e, enquanto for verdade, será feito uma lista de *Sets* do conjunto independente maximal, que será convertida para lista posteriormente. Então é feita uma lista chamada *subtraction_list*, para ver os elementos diferentes do conjunto independente maximal. Então é feita uma busca na *table_mark* que contém os elementos desta *subtraction_list* e é retornada uma lista (em *find_table*).

Questão 3.1:
Iniciando o algoritmo, colocamos uma lista de inteiros (*result*) iniciada com -1, que será o resultado da cor de cada vértice. Também temos a lista de booleanos temporária *available_color*, informando se a cor de cada elemento já foi colocada ou não. Depois é pego a lista de vizinhos de cada vértice com o método *grafo.grafo[u-1].vizinhos()*.