RELATÓRIO DE IMPLEMENTAÇÃO DA ATIVIDADE I

ALUNOS: Antônio Weingartner & Murillo Guidani

Questão 1: Para a representação computacional do grafo se optou por usar 4 estruturas de dados e 2 inteiros. Os inteiros contém as informações de quantos vértices e quantas arestas o grafo possui. As operações para obter o número de vértices e o número de arestas têm ambas complexidade de tempo O(1) devido a essa escolha de projeto. Para armazenar os graus de cada vértice se optou por usar uma lista. A posição k na lista representa o vértice k+1 e o valor na posição k indica o grau do vértice k+1. Para o armazenamento dos rótulos se optou também por usar uma lista da mesma forma que foi utilizada para armazenar os graus. O uso de uma lista é interessante pois permite que se obtenha o grau de um vértice em tempo constante. Para armazenar os vizinhos de um vértice se optou por utilizar uma lista de lista de vizinhos. A posição k da lista possui a lista de vizinhos do vértice k+1. Por último, se optou por usar um dicionário onde as chaves são conjuntos de 2 vértices e os valores são valores de ponto flutuante para armazenar os pesos das arestas. O uso do dicionário é interessante pois permite consultar o peso de uma aresta em tempo O(1) na maioria dos casos e O(n) no pior dos casos.

Questão 2: No algoritmo de busca em largura, foram utilizadas 4 estruturas de dados, 2 Sets do python, 1 dicionário e uma fila duplamente terminada (Deque do python). Os Sets foram utilizados para implementar 2 conceitos diferentes: o conceito de vértices explorados e o conceito de vértices expandidos. Na implementação os vértices expandidos são aqueles que já tiveram todos os seus vizinhos adicionados na fila e os vértices explorados são aqueles que já estão em algum nível da árvore de busca em largura. Optou-se por utilizar sets pois o teste para verificar se um set contém algum item tem complexidade de tempo computacional média de O(1) enquanto que a lista tem complexidade de tempo computacional média de O(n) para o mesmo teste. decidiu-se por usar uma fila duplamente terminada pois ela possui otimizações para inserção no início, enquanto que a lista padrão do python não possui tais otimizações. O dicionário é utilizado apenas para indexar as listas de vértices a cada nível da árvore de busca em largura.

Questão 3: Na implementação da questão 3 foram utilizados apenas sets e listas. A estrutura de dados que armazena as arestas que já foram visitadas foi implementada como um set. Isso pois, como já mencionado, a busca em um set é mais rápida que em uma lista.

Questão 4: Inicialmente decidiu-se estudar a possibilidade de implementação do Algoritmo de Dijkstra com heap binário e Fibonacci por razões de complexidade computacional. Em relação ao heap binário, as bibliotecas do python não possuem suporte adequado, pois a operação de decrease-key não é contemplada. Já com

relação a implementação com Fibonacci, as bibliotecas de terceiros encontradas no github não foram suficientes pois continham bugs. Nesse sentido, optou-se por uma implementação mais simples utilizando estruturas de dados análogas às representações estudadas durante a aula do dia 22 de novembro. Comparando com o Algoritmo de Dijkstra das anotações da disciplina, os elementos $\mathbf{D_v}$,$\mathbf{A_v}$ e $\mathbf{C_v}$ foram implementados como listas respectivamente **Distancia**, **Anterior** e **Visitados.** Sabe-se que não é a maneira mais eficiente de implementar esse algoritmo, mas por propósitos de demonstração entende-se que é suficiente.

Questão 5: Foram utilizadas listas nativas do python para representar a matriz D e D_k e suas respectivas operações. Ao testar com um pequeno grafo de teste tudo vai bem, porém em grafos grandes o algoritmo é lento devido a complexidade de tempo $O(|V|^3)$.