Universidade Federal de Santa Catarina - UFSC
Curso de Ciências da Computação - INE
Disciplina: Grafos
Alunos: Cristian A. Alchini, Leonardo Seishi Yamazaki, Victor do Valle Cunha

Relatório Atividade Prática A1

A atividade prática de construção de uma biblioteca de grafos e implementação dos algoritmos foi realizada pelo grupo utilizando a linguagem de programação Python. A criação da biblioteca que deu base para a implementação dos demais exercícios se deu a partir do uso de uma classe (denominada “Graph” no trabalho) que possui os métodos propostos pela atividade e dois atributos: “vertices” e “arestas”.

Estes dois atributos são as principais estruturas de dados da biblioteca. “Vertices” utiliza a implementação de dicionário do python e foi construído de modo que cada vértice de um determinado grafo seja uma chave. Os valores ligados a cada chave são outro dicionário, contendo uma chave denominada “rotulo” e outra chave denominada “vizinhos”. A primeira contendo um valor do tipo string e a segunda contendo um tipo set do python. O conjunto de vértices foi implementado dessa forma de modo a obter de forma rápida certas informações como o rótulo e os vizinhos do vértice. A complexidade média de uma operação de get() de um dicionário é O(1) e o pior caso amortizado é O(N). Foi considerado utilizar uma lista onde os vértices ocupariam um determinado índice, mas nesse caso, poderia ser um pouco mais difícil tanto lidar com índices ao invés de chaves quanto com uma possível troca no padrão do arquivo de entrada.

O atributo “arestas” é uma lista de tuplas, onde cada tupla contém três elementos: os dois vértices que compõem a aresta e o seu respectivo peso. As arestas foram modeladas dessa forma por ser uma implementação simples e oferecer acesso à aresta em tempo O(N) e informações como o tamanho do conjunto em tempo O(1).

Busca em largura

No algoritmo de busca em largura, para cada uma das três estruturas, vértices visitados (C), distância a partir da origem (D) e os antecessores (A). A opção por essa estrutura partiu, primeiramente, para facilitar as manipulações com essas estruturas, porque as instâncias fornecidas sempre iniciavam a contagem do vértices à partir do índice “1” e com o uso de uso de listas, que são indexadas à partir do índice “0, exigiriam um cuidado maior com seu uso. Além disso, como essas estruturas são dicionários e o Python os trata utilizando tabelas hash, então o custo da atribuição de valores é de complexidade O(1).

Ciclo Euleriano

No algoritmo de Hierholzer foi usado um dicionário para representar a estrutura de Arestas visitadas, para facilitar a procura das arestas na hora de testar, pois complicaria muito trabalhar com índices toda vez que procurar pela aresta entre os vértices vizinhos, principalmente nas condições de procurar se existe ou não alguma aresta pendente, foi feito uma função para auxiliar na criação desse dicionário onde a chave é a própria aresta. Em questão a complexidade o algoritmo fica em O(|E|), dependendo do número de arestas que terá que percorrer, apesar dos diversos loops ele não aumenta sua complexidade.

Floyd-Warshall

No algoritmo de Floyd-Warshall, foram utilizadas 2 matrizes construídas com um dicionário dentro de outro dicionário. As duas estruturas, Dold e Dnew, são construídas para armazenar os custos de cada aresta. Optou-se por utilizar apenas duas matrizes ao em vez de k+1, como proposto no algoritmo passado em sala, para se reduzir o custo de armazenamento computacional. Com relação ao a complexidade, assim como na busca em largura, o custo da atribuições são de O(1) e, com isso, o custo total é de $O(|V^3|)$.

Dijkstra:

Para realização do algoritmo de Dijkstra foram utilizadas estruturas que representam os vértices percorridos (C), a distância a partir da origem (D) e os antecessores (A). Os três foram modelados com o uso de dicionários, de modo a obter a informação de cada vértice com poucas linhas de código (ex. D[“vertice”]) e com complexidade média O(1) ou O(N) no pior caso. Não se usou o heap binário para representar as distâncias pela pouca familiaridade com a estrutura, mas se compreende a importância de estudá-la (bem como outras estruturas como o heap Fibonacci) de modo a obter melhor desempenho na execução do algoritmo.