Universidade Federal de Santa Catarina
Departamento de Informática e Estatística - INE
Curso de Ciências da Computação
Professor Rafael de Santiago

Relatório do Trabalho I de Grafos - INE5413

João Paulo Araujo Bonomo (21100133)
Gabriel Lima Jacinto (21202111)
Rodrigo Santos de Carvalho (21100139)

Florianópolis

2023

0.1 Representação

Para a representação de nosso **Grafo**, optamos pela representação em **lista de adjacências**, em que cada objeto *Node* tem um vetor de referências para as conexões com *Nodes* que ele alcança. Além disso, cada Conexão possui um peso associado, um *Node* de início e um de fim. No caso de **Grafos não-dirigidos**, o *Node* de início e o de fim estão invertidos para uma das referências, já que é possível caminhar de um ao outro independente da direção.

Optamos pela linguagem *C++* para o desenvolvimento do trabalho por conta da melhor orientação a objetos e da presença de recursos como ponteiros, que são ótimos para manter referência entre objetos, além de ser uma linguagem fortemente tipada, reduzindo problemas com tipos de variáveis, algo comum ao se trabalhar com **Grafos** em linguagens de tipagem dinâmica como *Python*.

0.2 Busca em Largura (BFS)

Para o desenvolvimento de nosso algoritmo de ***Busca em Largura (BFS)*** nos baseamos no **Algoritmo 3** da apostila da disciplina. Usamos da estrutura de *std::vector*s do C++ para representar os vetores *C*, *D* e *A*. Além disso, para termos uma referência para cada um dos *Nodes*, usamos um *vector* de ponteiros.

0.3 Ciclo Euleriano

Nosso ***Ciclo Euleriano*** foi baseado no **Algoritmo 5** da apostila da disciplina, e também conta com uma método auxiliar chamado *searchEulerianSubcycle*, fortemente inspirado no **Algoritmo 6**. Para essa implementação, usamos de *std::vector*s, de matrizes, ponteiros entre outras estruturas.

0.4 Algoritmo de Dijkstra

Usamos o **Algoritmo 11** da apostila para implementar o nosso próprio ***Algoritmo de Dijkstra*** em *C++*, com o adicional de que usamos uma estrutura de dados *Heap* para encontrar a menor distância atual em **tempo logarítmico**, melhorando significativamente a eficiência computacional de nosso algoritmo. Assim como os demais algoritmos e na prórpia representação do **Grafo** em si, usamos bastante o *std::vector* também.

0.5 Algoritmo de Floyd-Warshall

Assim como os demais algoritmos, baseamos-nos bastante na referência da apostila (neste caso, referimo-nos ao **Algoritmo 12**) para desenvolver nosso ***Algoritmo de Floyd-Warshall***. Esse algoritmo não exigiu nenhuma estrutura de dados muito complexa. Assim como os demais algoritmos, usamos de *std:vector*s e ponteiros.