UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CIÊNCIA DA COMPUTAÇÃO

Artur Luiz Rizzato Toru Soda

Relatório - Atividade Prática A1 - Grafos

Florianópolis
2024

1 INTRODUÇÃO

Este trabalho tem como objetivo aplicar os conhecimentos aprendidos na disciplina de Grafos até o momento, requisitando a implementação de um grafo e alguns algoritmos para serem aplicados sobre o mesmo para resolver problemas complexos.

2 EXERCÍCIO 1: REPRESENTAÇÃO

Neste exercício, para a representação do grafo não-dirigido e ponderado, foi implementado a classe *Grafo*, que possui os métodos requisitados: *qtdVertices(), qtdArestas(), grau(v), rotulo(v), vizinhos(v), haArestas(u,v), peso(u,v), ler(arquivo)*. Também foi pedido que essas funções, quando possível, tenham complexidade de tempo computacional O(1), e para isso foi-se utilizado a classe *Verticex* para para se ter um acesso mais fácil e rápido dos dados, cumprindo o objetivo pedido. Ademais, foi implementado, também o método *arestas()*, que retorna todas as arestas (u,v) existentes no grafo.

3 EXERCÍCIO 2: BUSCAS

Nesta tarefa, foi implementado o algoritmo de de busca em largura, que recebe como entrada um grafo e um vértice **S**, tendo como objetivo descobrir as distâncias de **S** dos demais vértices. Assim, no seu início é criado três vetores, um para representar os vértices visitados ou não, outro para armazenar as distâncias, e mais um para os ancestrais de cada um. Logo após, utilizando uma queue é iniciado um busca em largura, dos vizinhos de todos os vértices a partir de **S**, definindo então suas distâncias de **S** e ancestrais.

Sendo assim, para o cumprimento do exercício, foi utilizado o vetor de distâncias, onde a distância do vértice para o vértice S, representa o nível em que está.

4 EXERCÍCIO 3: CICLO EULERIANO

Para o exercício 3, foi implementado o algoritmo de *hierholzer***,** que busca por ciclos eulerianos, ou seja, um ciclo que passa por todas as arestas apenas uma vez, possuindo o seguinte funcionamento: Possui um vetor para controlar as arestas já visitadas ou não, e com a ajuda de uma função auxiliar (chamada de *busca()* nesta implementação*),* o algoritmo escolhe um vértice arbitrariamente (no código sempre é escolhido o primeiro), e vai "passando" de vizinho para vizinho de forma arbitrária (entretanto na implementação é selecionado a primeira aresta disponível), até voltar para para o vértice escolhido anteriormente (caso não seja possível voltar, o algoritmo retorna False, ou seja, não existe ciclo euleriano). Após isso, é escolhido alguma reta ainda não visitada, dentro dos vértices já passados, e procura outro ciclo euleriano a partir desse novo vértice, caso seja encontrado, esse novo ciclo é adicionado ao ciclo anterior, e então realizar esses passos recursivamente.

Sabendo disso, o algoritmo retorna False, caso haja alguma aresta que não foi visitada, ou se não foi possível formar um ciclo.

5 EXERCÍCIO 4: ALGORITMO DE BELLMAN-FORD OU DE DIJKSTRA

Neste exercício havia a opção de usar o algoritmo de Bellman-Ford ou de Dijkstra, pois os dois calculam os caminhos de custo mínimo, entretanto neste trabalho foram implementados os dois.

Em Bellman-Ford, tendo como entrada o grafo e um vértice **S**, ele passa por todos as arestas (u, v) qtdVertices() vezes, verificando se a "distancia de S para u" é maior que a "distância de S para v + o peso da aresta (u,v)", caso verdadeira é definido o novo ancestral e distância de v. Após isso é verificado a existência de ciclos negativos, retornando falso caso seja encontrado, se não, é retornado o vetor de distâncias mínimas e os ancestrais.

Agora em Dijkstra, além dos vetores de distâncias e ancestrais, também há um para controlar os vértices já visitados. Dessa forma, por ser um algoritmo guloso, a partir de um vértice **S**, passado como entrada, ele passa por todos os vértices, sempre pela aresta de menor peso, sem repetir vértices, para isso foi-se

implementado uma árvore binária, para uma melhor otimização na hora da escolha da aresta de menor valor. Assim, ao chegar a um novo vértice **v**, ele verifica todas as arestas (u,v) ligadas ao mesmo, se a “distância de S para v” é maior que a “distância de S para u + o peso de (u,v)”, caso verdadeiro, é definido a nova “distância de S para v”, seu ancestral, e a “distancia de S para u” é adicionado na heap. Dessa forma o algoritmo segue esses passos até que a binary heap esvazie ou todos os vértices forem conhecidos. Após todos os vértices serem visitados, é retornado as distâncias mínimas e os ancestrais.

6 EXERCÍCIO 5: ALGORITMO DE FLOYD-WARSHALL

Neste exercício foi implementado o algoritmo de Floyd-Warshall, que também resolve o problema de caminho de custo mínimo. Dessa forma, inicia uma matriz(i,j) para representar as distâncias de todos os vértices para os demais vértices do grafo, então, é passado por todos os valores da matriz (todas as distâncias de i para j), na ideia de inserir um vértice “k” entre um caminho pré-estabelecido de “u” para “v”, e define que a distância de u para v, é o mínimo entre “distância de u para v” e “distância de u para k + a distância de k para v”. Basicamente, ele verifica se existe outro caminho menos custoso de u para v. Após repetir esses passos em todos os valores da matriz k vezes (qtdVertices()) é retornado a matriz de distâncias.

7 CONCLUSÃO

Considerando todos os exercícios realizado, é possível concluir que o projeto proposto foi uma ótima prática para a aplicação de grafos e algoritmos aplicados nele, nos permitindo uma compreensão mais profunda das estruturas de dados e suas possibilidades, mostrando algoritmos de busca em largura, busca de ciclos eulerianos e o problema do caminho de custo mínimo.