CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LAYZA MARIA RODRIGUES CARNEIRO (1631378)

Teoria dos Grafos

Fortaleza

2023

Relatório

O atual trabalho consiste no desenvolvimento de uma biblioteca que implementa, em python, estrutura de dados para grafos orientados e não orientados.

Utilizei a **lista de adjacências** para ambas estruturas, visto que é mais eficiente em listar vizinhos e ocupa menos espaço.

Na tabela seguinte, N é o número de vértices, M o número de arestas e d é o grau máximo de um vértice.

	Lista de arestas	Matriz de Adj.	Lista de Adj.
Memória	$2M$	N^2	$2M$
Procurar aresta	M	1	d
Listar vizinhos	M	N	d
Adicionar aresta	1	1	1
Remover aresta	M	1	$2d$

Talvez a **matriz de adjacências** fosse mais interessante para analisar o caso teste, visto que, ela é uma opção melhor para grafos densamente conectados, mas a lista de adjacências é algo que tenho mais experiência e acredito que tenha sido uma escolha melhor, já que, seria preciso implementar as funções de dfs, bfs e busca de caminho mínimos, onde é preciso listar os vizinhos diversas vezes.

A maioria dos métodos foram implementados da mesma forma para grafos e dígrafos, com exceção da inserção de aresta, e o cálculo do grau de um determinado vértice.

Cada classe (grafo e digrafo) ao ser declarada inicia um dicionário chamado *lista_de_adjacência*, apesar de não ser uma lista, fica mais fácil de ser compreender ao chamá-la assim. E escolhi um dicionário, pois é mais fácil de acessar as conexões de cada vértice, assim como o peso da aresta.

Classe Grafo

Descrição da classe:

A classe *Grafo* implementa um grafo não direcionado e seus métodos básicos, utilizando uma lista de adjacência como forma de representar.

Métodos da Classe:

__init__(self):

Descrição: Construtor da classe *Grafo*.

Parâmetros: Nenhum.

Comportamento: Inicializa as estruturas de dados necessárias para representar o grafo.

adicionarVertice(self, vertice)

Descrição: Adiciona um vértice ao grafo, se ele ainda não estiver presente.

Parâmetros:

- *vértice*: Vértice a ser adicionado.

Comportamento: Verifica se o vértice já está presente no grafo e o adiciona à lista de adjacência, caso contrário, cria uma nova entrada na lista de adjacência para esse vértice.

adicionarAresta(self, u, v, peso=1)

Descrição: Adiciona uma aresta entre os vértices u e v, com um peso, caso não seja ponderado, é definido peso padrão = 1.

Parâmetros:

- *u*: Vértice de origem da aresta.

- *v*: Vértice de destino da aresta.

- *peso*: Peso da aresta (padrão = 1).

Comportamento: Adiciona uma aresta entre os vértices u e v na lista de adjacência, tanto de u para v quanto de v para u, se ainda não estiverem conectados.

numeroDeVertices(self)

Descrição: Retorna a quantidade de vértices existentes no grafo.

Parâmetros: Nenhum.

Comportamento: Conta e retorna o número de vértices presentes na lista de adjacência.

numeroDeArestas(self)

Descrição: Retorna a quantidade de arestas existentes no grafo.

Parâmetros: Nenhum.

Comportamento: Conta e retorna o número de arestas presentes na lista de arestas do grafo.

vizinhos(self, vertice)

Descrição: Retorna a vizinhança de um vértice específico.

Parâmetros:

- *vértice*: Vértice para o qual se deseja obter a vizinhança.

Comportamento: Retorna a lista de vértices adjacentes ao vértice especificado.

grauDoVertice(self, vertice)

Descrição: Retorna o grau de um vértice no grafo.

Parâmetros:

- *vértice*: Vértice para o qual se deseja calcular o grau.

Comportamento: Calcula e retorna o grau do vértice, ou seja, o número de vizinhos que o vértice possui.

pesoDaAresta(self, u, v)

Descrição: Retorna o peso de uma aresta entre os vértices `u` e `v`.

Parâmetros:

- *u*: Vértice de origem da aresta.

- *v*: Vértice de destino da aresta.

Comportamento: Procura e retorna o peso da aresta entre os vértices u e v, se essa aresta existir.

menorGrau(self)

Descrição: Retorna o menor grau presente no grafo.

Parâmetros: Nenhum.

Comportamento: Determina e retorna o menor número de conexões dos vértices no grafo, ou seja, o menor número de valores nas listas de adjacência.

maiorGrau(self)

Descrição: Retorna o maior grau presente no grafo.

Parâmetros: Nenhum.

Comportamento: Determina e retorna o maior número de conexões dos vértices no grafo, ou seja, o maior número de valores nas listas de adjacência.

bfs(self, vertice)

Descrição: Realiza a busca em largura a partir de um vértice específico no grafo.

Parâmetros:

- *vertice*: Vértice a partir do qual a busca em largura será iniciada.

Comportamento:

- Utiliza uma fila para explorar todos os vértices conectados ao vértice inicial, mantendo um conjunto de vértices visitados, uma lista de pais (mostrando o pai de cada vértice visitado) e um dicionário de distâncias (mostrando a distância a partir do vértice de origem).

- Percorre os vizinhos de cada vértice de forma nivelada, visitando todos os vizinhos diretos antes de avançar para os vizinhos mais distantes.

- Retorna os dicionários *pais* e *dist* que representam os pais de cada vértice visitado e a distância de cada vértice em relação ao vértice inicial, respectivamente.

dfs(self, vertice)

Descrição: Realiza a busca em profundidade a partir de um vértice específico no grafo.

Parâmetros:

- *vértice*: Vértice a partir do qual a busca em profundidade será iniciada.

Comportamento:

- Utiliza uma pilha para explorar todos os vértices conectados ao vértice inicial, mantendo um conjunto de vértices visitados, uma lista de vértices visitados na ordem em que foram visitados, uma lista de pais, e listas de tempo inicial e final de cada vértice.

- Percorre os vizinhos de cada vértice em profundidade, explorando o caminho o mais fundo possível antes de retroceder.

- Retorna listas contendo a ordem dos vértices visitados, os tempos iniciais e finais de cada vértice e o dicionário que representa os pais de cada vértice visitado.

bellman_ford(self, vertice)

Descrição: Utiliza o algoritmo de *Bellman-Ford* para procurar o caminho mínimo de um vértice para todos os outros no grafo.

Parâmetros:

- *vértice*: Vértice de origem a partir do qual se deseja encontrar os caminhos mínimos.

Comportamento:

- Inicializa listas de distâncias, vértices, e pais para realizar a busca pelo caminho mínimo.

- Executa um loop para relaxar as arestas, onde o número de vezes é igual a quantidade de vértices do grafo, comparando as distâncias e atualizando-as se necessário.

- Verifica a existência de ciclos negativos no grafo. Caso exista, informa que há um ciclo negativo e não retorna uma lista de distância e pais.

- Caso não existam ciclos negativos, retorna listas contendo os vértices, as distâncias mínimas e os pais de cada vértice.

dijkstra(self, vertice)

Descrição: Utiliza o algoritmo de *Dijkstra* para procurar o caminho mínimo de um vértice para todos os outros no grafo.

Parâmetros:

- *vértice*: Vértice de origem a partir do qual se deseja encontrar os caminhos mínimos.

Comportamento:

- Inicializa listas de distâncias e pais.

- Utiliza uma fila de prioridade para relaxar as arestas e encontrar os caminhos mínimos.

- Retorna listas contendo os pais de cada vértice e as distâncias mínimas em relação ao vértice de origem.

verticeMaisDistante(self, dist)

Descrição: Retorna o vértice mais distante do vértice desejado e sua distância.

Parâmetros:

- *dist*: Lista de distâncias mínimas dos vértices em relação ao vértice de origem.

Comportamento:

- Utiliza a lista de distâncias para encontrar o vértice mais distante e retorna o vértice e a sua distância.

Classe Digrafo

Descrição da classe:

A classe *Digrafo* implementa um grafo direcionado e seus métodos básicos, utilizando uma lista de adjacência como forma de representar.

Métodos da Classe:

A maioria dos métodos da classe digrafo são idênticos aos da classe grafo, com exceção dos que serão descritos a seguir.

adicionarAresta(self, u, v, peso=1)

Descrição: Adiciona uma aresta entre os vértices u e v, com um peso, caso não seja ponderado, é definido peso padrão = 1.

Parâmetros:

- *u*: Vértice de origem da aresta.

- *v*: Vértice de destino da aresta.

- *peso*: Peso da aresta (padrão = 1).

Comportamento: Adiciona uma aresta entre os vértices u e v na lista de adjacência, se ainda não estiverem conectados.

grauDoVertice(self, vertice)

- Descrição: Retorna o grau de um vértice no grafo, considerando tanto o grau de saída quanto o grau de entrada.

- Parâmetros:

  - *vértice*: Vértice para o qual se deseja calcular o grau.

-Comportamento:

- Calcula o grau de saída do vértice consultando o número de elementos na lista de adjacência referente ao vértice em questão.

- Calcula o grau de entrada do vértice percorrendo todos os vértices do grafo e verificando se há arestas direcionadas ao vértice especificado.

- Retorna a soma do grau de saída e do grau de entrada do vértice como o grau

Classe main

Descrição da classe:

A classe consiste em um script Python que realiza as seguintes operações:

lerArquivo(path)

Descrição: Função para ler um arquivo como o do caso teste e retornar uma lista de arestas.

Parâmetros:

- *path*: Caminho do arquivo a ser lido.

Comportamento:

- Lê o arquivo especificado linha por linha.

- Identifica linhas começando com "a" (indicando arcos/arestas) e adiciona à lista de arestas os vértices de origem, destino e o peso da aresta.

Retorno:

- Retorna a lista de arestas lidas do arquivo.

Execução Principal (`if __name__ == "__main__":`)

Comportamento:

- Inicializa um objeto da classe *Digrafo* vindo do módulo *biblioteca.digrafo*.

- Utiliza a função *lerArquivo* para obter a lista de arestas do arquivo ***USA-road-d.NY.gr***.

- Adiciona cada aresta lida ao digrafo utilizando o método *adicionarAresta*.

- Exibe informações sobre o grafo:

  - Quantidade de vértices.

  - Quantidade de arestas.

  - Grau mínimo e máximo no grafo.

- Calcula o vértice mais distante do vértice 129 e sua distância usando o método *dijkstra*.

Respostas do caso teste

a) O valor de G.mind = **1**

b) O valor de G.maxd = **8**

c) Um caminho com uma qtde. de arestas maior ou igual a 10 (apresentar a sequência dos vértices). = ***método não implementado corretamente***

d) Um ciclo com uma qtde. de arestas maior ou igual a 5 (apresentar a sequência dos vértices). = ***método não implementado corretamente***

e) O vértice mais distante (considerando os pesos das arestas) do vértice 129, e o valor da distância entre eles. **= vértice 90.644 e sua distância em relação a origem é 1.437.303**

Referências Bibliográficas

Além do material disponibilizado pelos professores (slides e livro) e conteúdo ministrado em sala de aula, utilizei para estudo e para tirar dúvidas:

- Algoritmos em Grafos (CodCad) | Neps Academy
- Grafos 01 - Uma Breve História de Grafos - NOIC