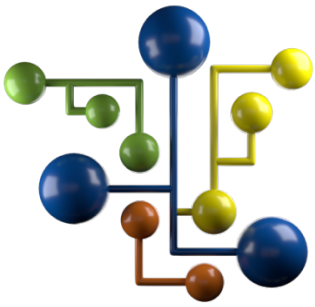

Data Science Academy

www.datascienceacademy.com.br

Introdução à Lógica de Programação

Nesta aula estudaremos o conceito de grafos (eu disse grafos e não gráficos).

Vamos definir o que são grafos e então, apresentarei um algoritmo usando grafos, chamado pesquisa em largura (do inglês breadth-first search, BFS).

Os grafos são amplamente usados em Inteligência Artificial e temos um curso completo sobre o tema aqui na DSA, chamado de Análise em Grafos Para Big Data, onde o BFS e outros algoritmos são estudados, bem como o banco de dados orientado a grafos, o Neo4j.

A análise de grafos também é usada na análise de redes sociais, a fim de buscar conexões. Através da análise de grafos temos diversas métricas que nos permitem coletar informações útil sobre os relacionamentos. A análise de grafos foi usada, por exemplo, nas investigações de lavagem de dinheiro, do episódio que ficou conhecido como Panamá Papers, para buscar conexões entre os envolvidos no esquema.

A pesquisa em largura é um dos muitos algoritmos usados na análise de grafos e permite encontrar o menor caminho entre dois objetos. Porém o menor caminho pode significar muitas coisas! Para exemplificar, é possível usar pesquisa em largura para:

• Escrever um algoritmo de inteligência artificial que calcula o menor número de movimentos necessários para a vitória em uma partida de damas.

• Criar um corretor ortográfico (o qual calcula o menor número de edições para transformar a palavra digitada incorretamente em uma palavra real; por exemplo, para modificar LEITOT -> LEITOR é necessária apenas uma edição).

• Encontrar o médico conveniado ao seu plano de saúde que está mais próximo de você.

Os algoritmos de grafos estão entre os algoritmos mais úteis e são aplicáveis a diversas situações. Há uma teoria por trás do conceito, chamada de Teoria dos Grafos, criada pelo matemático Leonhard Euler em 1736 enquanto tentava resolver o problema das Sete Pontes de Königsberg:

https://pt.wikipedia.org/wiki/Sete_pontes_de_K%C3%B6nigsberg

O Que é Um Grafo?

Um modelo de grafo é um conjunto de conexões. Por exemplo, suponha que você e seus amigos estejam jogando pôquer e que você queira descrever quem deve dinheiro a quem. Você poderia dizer “Bob deve dinheiro à Maria”.

O grafo completo poderia ser algo do tipo:

Bob deve dinheiro à Maria, Pedro deve dinheiro à Maria e a Marcos, e assim por diante. Cada grafo é constituído de vértices e arestas, como na figura abaixo.

E isso é tudo! Grafos são formados por vértices e arestas, e um vértice pode ser diretamente conectado a muitos outros vértices, por isso os chamamos de vizinhos.

Os grafos são uma maneira de modelar como eventos diferentes estão conectados entre si. Agora vamos ver a pesquisa em largura na prática.

Pesquisa em Largura (breadth-first search, BFS)

Nós já estudamos um algoritmo de pesquisa: a pesquisa binária.

A pesquisa em largura é um tipo diferente de algoritmo, pois utiliza grafos. Este algoritmo ajuda a responder a dois tipos de pergunta:

1: Existe algum caminho do vértice A até o vértice B?

2: Qual o caminho mais curto do vértice A até o vértice B?

Vejamos um exemplo muito elucidativo que foi extraído do livro "Grokking Algorithms: An illustrated guide for programmers and other curious people", do autor Aditya Y. Bhargava, o qual aliás recomendamos a leitura.

Vamos supor que você seja o dono de uma fazenda de mangas e esteja procurando um vendedor de mangas que possa vender a sua colheita. Você conhece algum vendedor de mangas no Facebook? Bem, você pode procurar entre seus amigos.

Essa pesquisa é bem direta. Primeiro, faça uma lista de amigos para pesquisar.

Agora vá até cada pessoa da lista e verifique se esta pessoa vende mangas.

Imagine que nenhum de seus amigos é um vendedor de mangas. Então, será necessário pesquisar entre os amigos dos seus amigos.

Cada vez que você pesquisar uma pessoa da lista, todos os amigos dela serão adicionados à lista.

Dessa maneira você não pesquisa apenas entre os seus amigos, mas também entre os amigos deles. Lembre-se de que o objetivo é encontrar um vendedor de mangas em sua rede.

Então, se Alice não é uma vendedora de mangas, você adicionará também os amigos dela à lista. Isso significa que, eventualmente, pesquisará entre os amigos dela e entre os amigos dos amigos, e assim por diante. Com esse algoritmo você pesquisará toda a sua rede até que encontre um vendedor de mangas. Isto é o algoritmo da pesquisa em largura em ação.

Encontrando o Caminho Mais Curto

Relembrando, existem dois tipos de pergunta que a pesquisa em largura responde:

1: Existe um caminho do vértice A até o vértice B? (Existe um vendedor de manga na minha rede?)

2: Qual o caminho mínimo do vértice A até o vértice B? (Quem é o vendedor de manga mais próximo?).

Você já sabe a resposta para a pergunta 1.

Agora, vamos tentar responder a pergunta 2. Você consegue encontrar o vendedor de mangas mais próximo? Por exemplo, seus amigos são conexões de primeiro grau e os amigos deles são conexões de segundo grau.

Você preferiria uma conexão de primeiro grau em vez de uma conexão de segundo grau, e uma conexão de segundo grau a uma de terceiro grau, e assim por diante.

Portanto não se deve pesquisar nenhuma conexão de segundo grau antes de você ter certeza de que não existe uma conexão de primeiro grau com um vendedor de mangas. Bem, a pesquisa em largura já faz isso!

O funcionamento da pesquisa em largura faz com que a pesquisa irradie a partir do ponto inicial. Dessa forma, você verificará as conexões de primeiro grau antes das conexões de segundo grau.

Você apenas segue a lista e verifica se a pessoa é uma vendedora de mangas. As conexões de primeiro grau serão procuradas antes das de segundo grau, e, dessa forma, você encontrará o vendedor de mangas mais próximo. Assim, a pesquisa em largura não encontra apenas um caminho entre A e B, ela encontra o caminho mais curto. Repare que isso só funciona se você procurar as pessoas na mesma ordem em que elas foram adicionadas.

Embora o BFS não seja a melhor opção para problemas que envolvem grafos grandes, ele pode ser empregado com sucesso em várias aplicações. Vou listar apenas algumas para ter uma ideia:

- Caminho mais curto de grafos não ponderados.
- Descobrir todos os nós alcançáveis a partir de um vértice inicial.
- Encontrar nós vizinhos em redes ponto a ponto como o BitTorrent.
- Usado pelos rastreadores nos mecanismos de pesquisa para visitar links em uma página Web e continuar fazendo o mesmo recursivamente.
- Encontrar pessoas a uma determinada distância de uma pessoa nas redes sociais.
- Identificar todos os locais vizinhos nos sistemas GPS.
- Pesquisar se existe um caminho entre dois nós de um grafo (localização do caminho).
- Permitir que pacotes transmitidos alcancem todos os nós de uma rede.

Aqui está o pseudocódigo para o BFS:

1. Adicione o vértice para iniciar a pesquisa pela primeira vez em uma fila vazia.

2. Marque esse vértice como visitado.

3. Extraia um vértice da fila e adicione seus vizinhos à fila, se não estiverem marcados como visitado.

4. Repita a etapa 2 até que a fila esteja vazia.

Vamos fazer a implementação em Python. Criaremos uma classe para construir o grafo e então executar o algoritmo BFS. Fique atento à identação e execute o script no Jupyter Notebook!

```
# Criação do Grafo e implementação do Algoritmo BFS

# Função para criar um grafo padrão
from collections import defaultdict

# Esta classe representa um grafo direcionado
# usando representação de lista de adjacências
class Grafo:

--# Construtor
--def __init__(self):

----# Dicionário padrão para armazenar o grafo
----self.graph = defaultdict(list)

--# Função para adicionar uma aresta ao grafo
--def addEdge(self,u,v):
----self.graph[u].append(v)

--# Função para o algoritmo BFS
```

```
--def BFS(self, s):

----# Marque todos os vértices como não visitados
----visited = [False] * (len(self.graph))

----# Crie uma fila para o BFS
----queue = []

----# Marque o nó de origem s como visitado e inclua na fila
----queue.append(s)
----visited[s] = True

----while queue:

------# Remova um vértice da fila e imprima
------s = queue.pop(0)
------print (s, end = " ")

------# Obter todos os vértices adjacentes do vértices desenfileirados.
------# Se um adjacente não foi visitado, marque-o visitado e inclua na fila
------for i in self.graph[s]:
--------if visited[i] == False:
----------queue.append(i)
----------visited[i] = True



# Cria uma instância da classe, construindo um grafo com diversas arestas
g = Grafo()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

# Executa o algoritmo BFS
print ("Aqui está o caminho a seguir para atravessar o grafo (começando do vértice 2):")
g.BFS(2)
```

Aqui você encontra outros exemplos para praticar:

https://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/

Até a Aula 10!

#aula09