**Material de Apoio**

[**https://drive.google.com/file/d/16HvsVeOuK2xs0FID\_BZvkU-HftyXGVPc/view?usp=sharing**](https://drive.google.com/file/d/16HvsVeOuK2xs0FID_BZvkU-HftyXGVPc/view?usp=sharing)

[**https://drive.google.com/file/d/1HweLgEm3baV935x6udA7tuVUlMDlU\_Qo/view?usp=sharing**](https://drive.google.com/file/d/1HweLgEm3baV935x6udA7tuVUlMDlU_Qo/view?usp=sharing)

[**https://drive.google.com/file/d/1\_7jpYwyqDnwfvLKbWBp\_sc7IptywDXO1/view?usp=sharing**](https://drive.google.com/file/d/1_7jpYwyqDnwfvLKbWBp_sc7IptywDXO1/view?usp=sharing)

**Teoria dos Grafos**

**(G= V,E)**

\* **G** é um conjunto de vértices **V** (também chamadas de nós) que são conectados por arestas ou arcos **E.**

**Então, G = (V(G), E(G))**

-> É uma estrutura matemática composta por dois conjuntos sendo:

**V(G)=** um conjunto de elementos que são chamados de vértices,

**E(G)=** um conjunto de pares de elementos **V(G)** cada par é chamado de aresta.

Assim temos que vértices **adjacentes** são vértices que são ligados por aresta e **incidentes** são arestas que ligam os vértices.

-> São estruturas formadas por um conjunto de vértices (pontos ou elementos) não vazios e por um conjunto de arestas (retas ou linhas) que fazem alusão a uma relação binária. Assim com grafos podemos representar elementos e suas possíveis relações entre objetos a um determinado conjunto. Grafos evidenciam conexões de problemas do mundo real de maneira eficaz tendo suas múltiplas possibilidades de soluções analisadas em profundidade.

**Modelos de Grafos**

(Considerando as partes de um grafo: vértices e arestas)

- *Grafos de Comunicação*: *vértice -* computadores, satélites, telefonia. Arestas - cabos de rede, fibra ótica, onda de rádio.

- *Grafos de Circuitos Elétricos*:  *vértice* - portas lógicas, registradores, processadores.  *Arestas* - filamentos.

- *Grafos de Sistemas Hidráulicos*:  *vértice* - reservatório de água, estações de bombeamento. *Arestas* - tubulações.

- *Grafos do Sistema Financeiro*:  *vértice* - ações, moeda.  *Arestas* - transações.

- *Grafos do Sistema de Transporte*:  *vértice* - aeroportos, estradas, rodovias.  *Arestas* - estradas, ruas, rodovias, vias aéreas.

- *Grafos da Internet*:  *vértices* - páginas web. *Arestas* - hiperlinks.

- *Grafos das relações* pessoais: ­ vértices *-* pessoas atores.  *Arestas* - amizades, trabalho conjunto em filmes.

**TIPOS DE GRAFOS**

**Grafos Simples:** reconhecido por não ter laços (pontas iniciais e final do arco coincidentes), nem mais de uma aresta conectando dois vértices (arestas múltiplas). São conhecidos quando seus vértices são isolados, ou seja não são adjacentes com outros vértices. Também não são direcionados e suas arestas não são paralelas.

V = {P1, P2, P3}

E = {{P1, P2}}

\* Na representação os vértices P1, P2 e P3 estão conectados somente com as arestas P1 e P2. Como não há conecção com o vértice P3 ele não aparece no conjunto de elos.

**Grafos Regular:** elaborado de tal maneira que cadas vértice passa a ter o mesmo número de adjacências. Além disso, cada vértice passa a ter o mesmo número de ligações (graus), também chamado de valência.

- *Grau* se refere a quantidade de arestas que se conectam a um vértice.

**Grafos Vazios:** conhecido como grafo nulo, é conhecido por não ter arestas em sua composição possuindo apenas vértices.

V = {A, B, C, D}

E = { }

**Grafos Trivial:** reconhecido por possuir um único vértice e não ter arestas em sua estrutura.

V = {1}

E = ∅

|V| = 1 e |E| = 0

**Grafos Direcionado (digrafo):** reconhecido por ter um sentido (direção). Logo se temos as arestas Vi e Vj uma é conectada a outra por meio de uma linha, com uma seta saindo de Vi e apontando para Vj.

**Grafos Multigrafo não Direcionado (sem sentido de direção):** possui arestas múltiplas que conectam com os mesmos extremos. Um Grafo **G-(V,A)** é definido como multigrafo quando em sua estrutura há duas ou mais arestas entre pares de vértices de **G.** Assim, é reconhecido por ter arestas paralelas.

**Grafos Bipartidos:** reconhecido por permitir que seu conjunto de vértices, representado por **V,** seja patrocinado em dois subgrafos (V1 e V2), de modo que todo arco (aresta) do grafo conecta o vértice do primeiro grafo ao vértice do segundo grafo.

V = V1 V2 tais que V1 ∩ V2 = ∅

indica que V1 e V2 são grafos subconjuntos e a intersecção de ambos é igual a vazio.

**Grafos de Ciclo Eureliano (trilha eureliana fechada)**: quando há ciclo em G que englobe todas as arestas de G. Do mesmo modo, uma trilha fechada eureliana é um caminho em que se passa uma única vez por cada aresta do grafo. Tendo os vértices que o compões de grau par. (Ex. pag 5 apostila 1).

**Grafos de Ciclo Hamiltoniano:** referen-se aos grafos que possuem um ciclo em que todos os seus vértices estão inclusos, repetindo-se apenas o primeiro vértice de onde se iniciou o ciclo. (Ex. pag 6 apostila 1).

**Grafos Isoformos:** grafos que contém o mesmo número de vértices e o mesmo número de arestas (elos). Além disso, o vértice do primeiro grafo deve estar na mesma posição que o vértice do segundo grafo, ou seja, devem ser correspondentes.(Ex. pag 6 apostila 1).

**REPRESENTAÇÕES GEOMÉTRICAS DOS GRAFOS**

Exemplos páginas 8 a 11.

**DIFERENÇAS ENTRE GRAFOS, MATRIZ DE ADJACÊNCIA E LISTA DE ADJACÊNCIA.**

**->** Apesar de muito eficazes visualmente, grafos representados por pontos e linhas, não podem ser lidos por computador, assim, usamos matrizes de adjacências, que permitem essa interação.

**\*** *Matrix de Adjacência:* desenvolvida para representar um grafo em uma modelagem computacional direcionando os vértices por linhas e colunas. Se houver vínculo entre os vértices adiciona o valor 1 na matriz. Se não houver adiciona-se o valor 0.

Assim se em uma representação geométrica dos grafos temos:

- V1 se relacionando diretamente com V2, V5 e V6.

- V2 se relacionando diretamente com V1, V4 e V5.

- V3 se relacionando com V4.

- V4 se relacionando com V2, V3,V5 e V6.

- V5 se relacionando com V1, V2 e V4.

- V6 se relacionando com V1 e V4.

Temos a seguinte matriz de adjacência:

Obs: fazer tabela substituindo O e 1 onde não há arestas. (Pag 12 apostila 1).

\* *Listas de Adjacências*: é uma lista que exibe as ligações de todos os vértices.

- ver ex pagina 12.

**Obs**: *se o grafo for direcionado, ou seja, se suas arestas possuírem sentido de ligação, é considerado vizinho de um vértice apenas aquele que tiver a aresta com sentido ao próximo vértice.*

*Ex.:* se o vértice V possuir uma aresta direcionada para o vértice X e ao mesmo tempo houver uma aresta **oriunda** do vértice U apontando para o vértice V, será considerado vizinho do vértice V somente o vértice X.

**DATA MINING E GRAPH MINING**

Grafos são *estrutura de dados* compostas de um conjunto de vértices, que representam elementos de dados, e um conjunto de arestas, que representam as associações entre esses vértices. Grafos podem ser representados graficamente ou matematicamente pelo elemento de dois conjuntos:

G:

V < {0,1,2,3}

A < {(0,1), (0,2), (1,3), (1,2), (2,3)}.

-> Para usar grafos no cotidiano pessoal é preciso aplica-lo em uma instrução de linguagem.

**- Como construir Grafos em Python:** usando um módulo chamado NETWORKX que tem várias funções para criação, manipulação e até o desenho de grafos.

**Script:**

**O script é este:**

**1. import networkx as nx**

**2. G = nx.Graph()**

**3. G.add \_ node('A')**

**4. G.add \_ node('B')**

**5. G.add \_ node('C')**

**6. G.add \_ edge('A', 'B')**

**7. G.add \_ edge('B', 'C')**

**8. G.add \_ edge('C', 'A')**

**9. nx.draw(G, with \_ labels=True)**

- LInha 1: importar o networkx, para ser usado no código.

- Linha 2: criar o grafo G - inicialmente vazio está sem vértices - chamados *nodes.*

- Linha 3,4 e 5: função *add* para adicionar os vértices.

- Linha 6,7,8: função *add* para adicionar as arestas.

- Linha 9: desenha o grafo por meio da chamada do comando nx.draw (**ver pag 4 apostila 2).**

**Formas de Grafos por Meio de Classes em Python**

**1: Graph:** grafo não direcionado, que ignora arestas múltiplas entre vértices (por exemplo, duas arestas entre A e B), mas aceita loops (aresta de um nó para ele mesmo);

**2: DiGraph:** grafos direcionados (com operações específicas para esse tipo de grafo);

**3: MultiGraph:** aceita arestas múltiplas (não direcionadas) entre dois vértices.

**4: MultidiGraph:** versão direcionada do Multigrap.

Obs:

- Hashable: admite strings, tuplas, inteiros, etc...

- Todos as classes de grafos permitem como nó de arestas qualquer objeto hashable.

- No intuito de simplificar prefere-se criar grafos apenas fornecendo ***listas*** de arestas.

Ex.:

1. import networkx as nx

2. arestas = [('Brasilia','Lisboa'), ('Lisboa','Paris')]

3. G = nx.Graph()

4. G.add \_ edges \_ from(arestas)

5. nx.draw(G, with \_ labels=True, node \_ size=2800)

PARIS

**LISBOA**

BRASILIA

**GRAFOS A PARTIR DE BANCO DE DADOS RELACIONAIS**

**Graph Mining:** grafos são obtidos a partir de bando de dados reais. Para a criação do grafo em Banco de Dados Relacionais a partir da listak retornada pelo objeto cursor usa-se o *comando “add\_weihted\_edges-from”.*

**GRAFOS A PARTIR DE BANCO DE DADOS NÃO RELACIONAIS**

**-** Utiliza-se o módulo ***pymongo*** (pag 10 e 11 apost 2).

**CLASSIFICAÇÃO DE GRAFOS**

- São classificados de acordo com seus vértices e arestas, levando-se em consideração a quantidade, a disposição desses elementos e o padrão de ligação formado.

-> **Laço** (loop): uma aresta que tem origem e destino no mesmo vértice. O comando *ping localhost* é responsável pelo envio dos dados cuja origem e destino do mesmo vértice.

-> **Arestas Múltiplas:** são arestas múltiplas (multígrafos) que **possuem** o mesmo par de arestas de origem e destino.

Ex. Uber onde há diferença de horários entre mesmos itinerários.

-> **Pseudografos:** apresentam laços e arestas múltiplas.

-> **Simples:** não possui loop ou arestas múltiplas.

-> **Direcionado:** suas arestas representam a direção de arestas entre dois vértices quaisquer.

Ex.: Twiter – quando não há reciprocidade entre seguidor e seguidores.

**TERMINOLOGIA DOS GRAFOS**

**-** Trata-se de orientações sobre **direcionamento** de arestas, permissão de arestas e possibilidades de laços.

|  |  |  |  |
| --- | --- | --- | --- |
| Tipos | Arestas | Arestas Múltiplas Permitidas? | Laços Permitidos? |
| Grafo Simples | Não Orientada | Não | Não |
| Multigafo | Não Orientada | Sim | Não |
| Pseudografo | Não Orientada | Sim | Sim |
| Grafo Orientado Simples | Orientada | Não | Não |
| Multigrafo Orientado | Orientada | Sim | Sim |
| Grafo Misto | Orientada e Não Orientada | Sim | Sim |

**\* Definições (Rosen – pág. 598 a 600) sendo {u,v} extremidades de uma aresta associada:**

**1- Incidência:** quando aresta “e***”*** se conecta a dois vértices {u,v} não orientado é denominado que “***e”*** é incidência dos vértices {u,v}. -

**2- Grau de Vértice não orientado:** *é o* número de arestas incidentes a ele, exceto que um laço em um vértice contribui duas vezes ao grau daquele vértice. Ex. o grua do vértice ***v = gr(v).***

**3- Sendo** (u,v) uma aresta em Grafo **G,** com arestas orientadas, então ***u*** é adjacente para ***v e v*** é adjacente para ***u.*** Então temos que o vértice ***u*** é inicial a {u,v} e o vértice ***v*** é final ou (terminal) de {u,v}.

Obs: os vértices final e terminal de um laço são os mesmos.

**4- Sendo** um grafo com arestas orientadas, o grau de entrada de um vértice ***v***, sendo gr-(v), é o número de arestas que tem ***v*** com seu vértice final. O grau de saída de ***v,*** indicado por gr(v), é o número de arestas que tem ***v*** como seu vértice inicial.

Obs: um laço em um vértice contribui tanto para o grau de entrada para quanto o grau de saída.

***5- Vértices Adjacentes (vizinhos):*** são vértices u,v não orientados chamados de extremidades de uma aresta em G em um grafo.

***GRAFO COMPLETO***

- Quando o grafo é simples (sem laços e arestas múltiplas) e cada vértice é adjacente a todos os outros.

***GRAFO CÍCLICO***

- Quando o grafo produz ***ciclo,*** se ele possui um conjunto de pelo menos três arestas em que o vértice de origem da ***sentido*** a primeira aresta é o vértice de destino da última aresta segue o mesmo sentido.

***GRAFO ACÍCLICO***

***-*** Quando o grafo não possui qualquer ciclo.

***GRAFOS BIPARTIDOS***

***-***  Quando podemos separar dois conjuntos de vértices desse grafo, em que os vértices de um conjunto só se relacionam com os vértices do outro conjunto.

**\* Atribuição de grafos bipartidos:** *pode ser usado para modelagem onde atribui tarefas a pessoas.*

**PASSEIOS, CAMINHOS E CICLOS DE GRAFOS**

**(PASSEIO DE GRAFOS-WALK)**

* Um **passeio** em um grafo é uma sequência de vértice em que cada vértice é adjacente (vizinho) do outro.

- *Passeio Fechado*: se o primeiro vértice da sequência coincide com o último.

***->*** Um passeio sem arestas repetidas é um caminho **path** em um grafo.

***-> Ciclo*** é um caminho fechado, ou seja, não tem arestas repetidas e o vértice de origem é o mesmo de origem é o mesmo de destino.

**5**

**4**

**6**

**3**

**0 2**

**CIC**

1

* **Exemplificando Passeio Aberto** (Walk): consta na sequência de vértices 0-3-4-5-6. Nesse passeio (não há repetição de arestas) é um caminho simples, pois não foi repetido nenhuma aresta ou vértice. Mas ele não é fechado (pode ser chamado de aberto), pois o primeiro vértice é diferente do último.
* **Exemplificando Passeio Fechado** (Walk): consta na sequência de vértices 1-2-3-4-3-2-1. É dito fechado porque sai do vértice 1 volta para ele. Porém ao repetir arestas **não é** considerado um caminho.
* **Exemplificando Caminho não simples:** consta na sequência 3-6-4-5-6-0 é um caminho pois não repete arestas, mas como passa mais de uma vez no vértice 6, não é considerado simples.
* **Exemplificando Ciclo não simples:** consta na sequência 0-3-6-4-5-6-0 um ciclo, porém, não pode ser considerado simples pois repete o vértice 6.

**CICLO E CAMINHO EURELIANO EM UM GRAFO (G)**

Um ciclo e um caminho eureliano em G é um caminho e um ciclo simples que contém todas as **arestas** de G.

**CAMINHO E CICLO HAMILTONIANO EM UM GRAFO (G)**

Um caminho e um ciclo hamiltoneano em um grafo G quando o caminho e o ciclo passam por todos os **vértices** exatamente uma vez.

**OBS.:** Note que eureliano tem como referência as arestas e Hamilton tem como referência as vértices.

* **Exemplificando:**

- Consta na sequência 0-1-2-3-4-5-6 **caminho** hamiltoniano pois não passa por todos os vértices e uma só vez.

- Consta na sequência 0-1-2-3-4-5-6-0 **ciclo** hamiltoniano mas não possui um caminho ou ciclo eureliano, ou seja, não é possível percorrer todas as arestas sem repetir.

**GRAFOS PONDERADOS**

**(Determinam o menor caminho entre dois vértices com menor custo total)**

* São grafos com valores chamados de **pesos** (weighted) atribuídos a suas arestas. São utilizados para representar uma grande quantidade de cenários.

**Exemplos**:

1. Esquema de roteamentos de redes em que cada vértice representa um roteador, e as arestas, os links de comunicação entre eles. O -“***PESO”*** das arestas pode representar a velocidade de conexão ou a carga de tráfego naquele link.
2. Rota de entrega de produtos em que os vértices são os locais de entrega e as arestas a distância de entrega entre essas localidades. Sendo a unidade de medida de distância o ***peso.***
3. Sistema de atribuição de tarefas sendo os vértices representados por pessoas e tarefas a serem realizadas (grafos bipartidos) e as arestas representam o tempo dedicado por uma pessoa na realização de uma tarefa.

**ALGORITMO DE DIJKSTRA**

A partir da escolha de um vértice de origem o **algoritmo** calcula o custo mínimo desse vértice para todos os demais vértices do grafo. Pode ser utilizado para grafos orientados (dígrafos) *ou não* e admite que todas as arestas tenham pesos negativos.**(ver algoritmo de Distra pág 17).**

**GRAFOS PLANARES**

Quando o Grafo é desenhado em um plano sem quaisquer arestas se cruzarem.

**Obs**.: cruzamento de arestas é a interseção de restas ou arcos que as representam em um ponto diferente de sua extremidade comum.

**Representação Planar de Grafo de Euler**

**(r = e-v + 2)**

Euler mostrou que a representação planar de um grafo simples divide o plano no mesmo número de regiões.

Temos: ***r*** = regiões; ***e***= arestas e ***v***= vértices.

* **Representação de Grafos Planares:**

**-** Se G é um grafo simples conexo planar, com **e** arestas e **v** vértices, sendo **v >- 3,** então **e -<** 3v – 6.

**-** Se G é um grafo simples conexo planar, então G tem um vértice de grau menor igual a 5. (grafos cruzados e planares fig 13 pag 19).

-> Aplicações de Grafos planares: projetos de circuitos integrados, projetos de circuitos impressos, projetos de estradas conectando cidades, projetos de linhas de transmissão elétrica.

**APLICAÇÃO DAS TÉCNICAS DE CLASSIFICAÇÃO DE GRAFOS EM PYTHON**

**(NETWOKX)**

O módulo Python NETWORKX permite aplicar os conceitos e as técnicas de classificação retromencionadas.

Exemplo em Grafos Script de Grafo Completo de 5 Vértices:

1. import networkx as nx
2. G = nx.complete\_graph (5)
3. nx.draw (G, with\_labels=True, node\_size=2800)

* método ***complete\_graph*** gera grafo completo de ***n*** vértices.

Seguindo exemplo acima segue a execução do script acima:

**1**

2 3

4

0

* Atributos ***nodes*** E ***edges:*** são atributos que contém a lista de vértices e de arestas do grafo.-
* Atributo **degree** retorna uma lista de tuplas em que o primeiro elemento é o valor do vértice e o segundo seu respectivo grau.
* ***Grau*** de vértices são número de arestas que saem de cada vértice. No exemplo retro mencionado temos grau 4 em todos os vértices por se tratar de um grafo completo.
* Comando ***all\_simple\_path*** gera uma lista com todos os caminhos simples entre dois vértices, informando o Grafo (vértice de origem – **source** e vértice de destino- **target**).

1 import networkx as nx

2 G = nx.complete\_graph (5)

3 for path in nx.all\_simple\_path(G, source = 0 target = 3):

4 print(path)

5 nx.draw(G, with\_labels=true, node\_size=2800) resultado da execução pag 20

* Comando ***eurelian­\_circuit:*** retorna um circuito eureliano para o grafo desejado.
* Comando ***is\_eurelian*** retorna um valor booleano se o grafo em questão é eureliano. Em um circuito eureliano, todos os vértices do grafo são percorridos uma única vez no caminho. Assim o circuito começa em um vértice percorrendo uma vez só o caminho e retorna para o vértice de origem.

O networkX contém função para cálculos de menores caminhos em um grafo:

Utilizando o arquivo de texto *sp.edgelist.* defini-se a estrutura do GRAFO. Assim, dentro desse arquivo a lista de arestas com respectivos ***pesos*** (weights) que devem ser inseridas no Grafo.

Obs: o atributo (weigth) armazena o peso das arestas em questão. Então, aplica-se o algoritmo de Dijkstra para encontrar o caminho de menor custo após definir vértice de origem e destino enfim ***imprimi-se*** o custo total.

* **Grafos são excelentes para potencializar Data Mining.**
* Leituras recomendadas BATHIA, P. Data mining and data warehousing. Cambrige: Cambridge University Press, 2019. DRABAS, T. Practical data analysis cookbook. Birmighan: Packt, 2016. LAYTON, R. Learning data mining with Python. 2nd ed. Birmighan: Packt, 2019. PLATT, E. L. Network science with Python and NetworkX quick start guide. Birmighan: Packt, 2019.

**ESTRUTURA DE DADOS PARA AI II**

**Fundamentos sobre Árvores Binárias:** é a representação de nós e arestas que se interligam a um elemento inicial, chamado raiz. Essa interligação entre os nós forma a árvore e as folhas. O nó superior a folha será o nó pai que se liga aos nós folha.

RAIZ

PAI PAI

FOLHA

FOLHA FOLHA FOLHA FOLHA

**TIPOS DE ÁRVORES**

**ÁRVORES BINÁRIAS: s**ão estruturas de dados hierárquicas, no qual todos os nós têm grau 0,1 ou 2 por nó.

**ÁRVORE ESTRITAMENTE BINÁRIA:** são estrutura de dados hierárquicas, na qual todos os nós têm os graus 0,2.

**ÁRVORES BINÁRIAS COMPLETAS:** são estrutura de dados hierárquicas no qual todas as folhas estão no mesmo nível.

**TIPOS DE ÁRVORES EM PYTHON**

Primeiramente implementa-se um conjunto de nós, sendo que cada nó será um objeto com uma chave, um valor e uma referência aos seus filhos, que podem estar do lado direito ou do lado esquerdo da árvore. Depois cada chave servirá para identificar um nó e o valor representará o que o nó armazena. Assim, haverá nós com valores numéricos.

* **Nó em Python:** é definido por meio da classe ***binary Search tree node (BSTNode)***

01 >>> class BSTNode (object):

02 >>> def \_ int \_ (self, key, value=None, left=None, right=None):

03 >>> self.key = key

04 >>> self.value = value

05 >>> self.left = left

06 >>> self.right = right

**Campos**:

***left e right*** usa-se como referências para outros nós

***key*** será as chaves de identificação dos nós

***value*** será o valor armazenado neles.

01 >>> root = BSTNode (4)

02 >>> root.left = BSTNode (1)

03 >>> root.right = BSTNode (9)

04 >>> root.left.left = BSTNode (2)

* Entendimento: implementa-se a criação do nó raiz com a chave (4), sem um valor armazenado e adiciona-se dois filhos a esquerda (1) e na direita (9), bem como um filho ao nó da esquerda (1) de valor (2).

Exemplo de uma árvore binária utilizando Python:

9

1 4

**2**

**ALGORITIMO DE BUSCA EM LARGURA**

**(BSF)**

**BSF** se originou da teoria dos Grafos, com a função de realizar buscas em estruturas de dados com árvores e grafos, inicializando a partir do nó raiz e continuando a verificação dos nós vizinhos até encontrar o elemento desejado buscado. Examina a busca desinformada e exaustiva em todos os nós de uma árvore ou arestas/vértices até encontrar o elemento buscado.

**Finalidade:** realizar a exploração de árvores e grafos a fim de buscar determinados elementos a ser encontrados. Assim garante que cada nó ou grafo de uma árvore não será visitado mais de uma vez, agrupando os nós visitados em estrutura de dados em fila (FIFO) a fim de assegurar que não haverá dupla verificação. (Ex pag 8, apostila 3).

Ao se aplicar BFS em estrutura de dados em uma árvore ou grafo há uma fila de verificação de todos os nós que serão analisados na busca, à medida que um é verificado, sai da fila e os próximos elementos entram e são verificados.

- A estrutura de dados em nós ***GRAFOS:*** tem suas arestas interligadas pelos nós que se conectam entre si.

- A estrutura de dados em nós ***ÁRVORES:*** possui os mesmos nós se interconectando com os nós pais, formando nós pais e folhas.

-> Assim toda árvore é um **grafo**, mas nem todo grafo será uma **árvore**. (Ex figura 10, pag 9, apostila 3).

Em Python teremos:

01 >>> graph = { 'A': set (['B', 'C']),

02 >>> 'B': set (['A', 'D', 'E']),

03 >>> 'C': set (['A', 'F']),

04 >>> 'D': set (['B'],

05 >>> 'E': set (['B', 'F']),

06 >>> 'F': set (['C', 'E'])}

* Implementa-se os grafos com conexões adjacentes entre os elementos a,b,c,d,e, f. Depois executa-se o algoritmo de busca em largura (BFS)

01 >>> def bfs (graph, start):

02 >>> visited, queue = set (), [start]

03 >>> while queue:

04 >>> verrtex = queue.pop (0)

05 >>> if vertex not in visisted:

06 >>> visited.add (vertex)

07 >>> queue.extend(graph[vertex] – visited)

08 >>> return visited

09 >>> bfs(graph, 'A') # {'B', 'C', 'A', 'F', 'D', 'E'}

* Linhas 1 e 2: algoritmo BFS por dois parâmetros (grafo e início), em seguida atribui-se o método ***set*** uma fila (queque). Então coloca-se uma condição para iniciar a ***fila (while),*** na qual enquanto o vértice (***vertex)*** for igual a ela se deve retirar o elemento (pop) já verificado em (***0).***
* Linhas 5 e 6: pag 11