Programação Python

Aula 11: Introdução aos Grafos

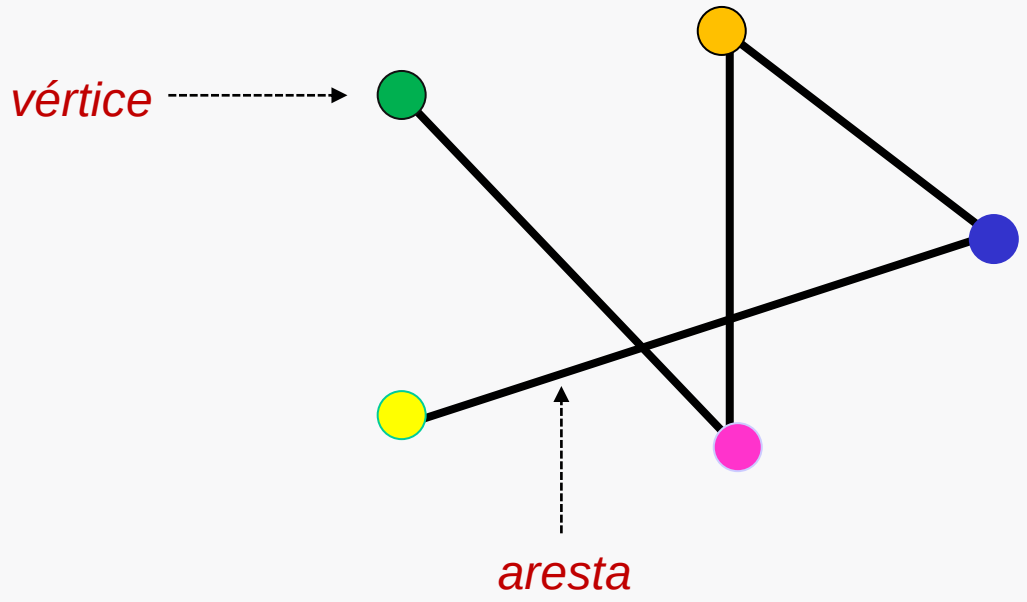
Prof. Eduardo Corrêa Gonçalves

31/03/2021

Sumário

Introdução (1/9)

- **O que é Grafo?**
  - ED formada por um conjunto de **vértices** e um conjunto de **arestas**.
    - **Ex.**: grafo com 5 vértices e 4 arestas.



  - Vértices também são chamados de **nós** e arestas de **arcos**.

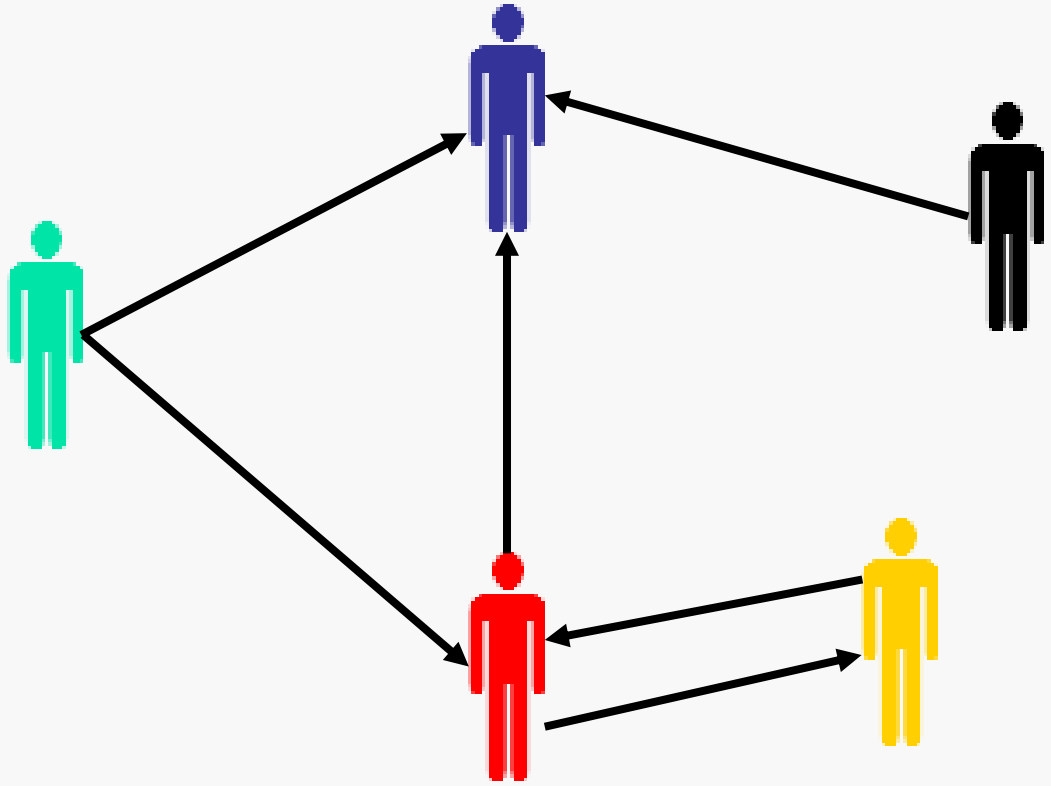
Introdução (2/9)

- **Por que Estudar Grafos?**
  - Importantíssima **ferramenta matemática** com aplicação em diversas áreas do conhecimento
  - Associando-se **significados** aos **vértices** e às **arestas**, o grafo passa a constituir um modelo de um problema real.
  - Existem centenas de problemas que empregam grafos com sucesso como veremos a seguir.

Google Knowledge Graph
https://youtu.be/mmQl6VGvX-c

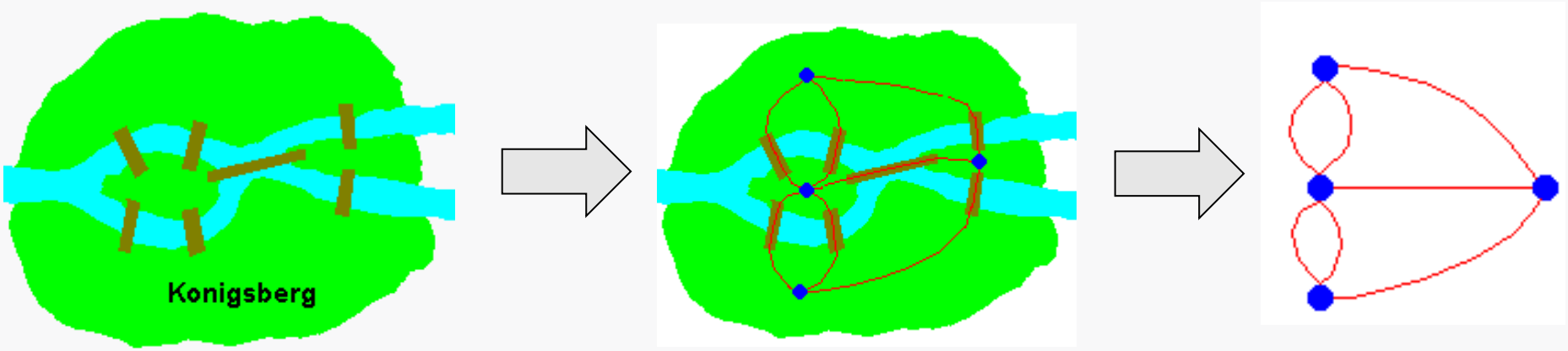
Introdução (3/9)

- **Aplicações – Modelagem de Redes Sociais**
  - Vértice = pessoa
  - Há uma aresta do vértice A para o B caso a pessoa A siga a pessoa B



Introdução (4/9)

- **Aplicações – Representação de Cidades / Ruas / Estradas / etc.**
  - **Ex.**: **problema de Königsberg**
    - Em Königsberg, Alemanha, um rio que passava pela cidade tinha uma ilha. Logo depois de passar por essa ilha se bifurcava em 2 ramos. Na região existiam 7 pontes.
    - É possível andar por toda cidade de tal modo que cada ponte seja atravessada exatamente uma vez?



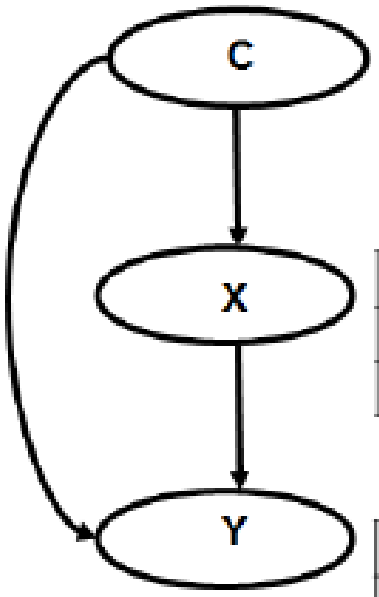
- A resposta é **NÃO** (*detalhes em Nogueira, 2019*)

Introdução (5/9)

- **Aplicações – Coloração de Mapas**
  - Quantas cores são necessárias para colorir o mapa do Brasil, sendo que estados adjacentes não podem ter a mesma cor?
    - Vértices = estados. Arestas apenas entre estados que fazem fronteira.

Introdução (6/9)

- **Aplicações – Redes Bayesianas**
  - Na ciência de dados, são inúmeras as aplicações dos grafos. Por exemplo, uma redes bayesiana representa variáveis e suas dependências em um **grafo direcionado acíclico**.



P(C=0)	P(C=1)
7/14	7/14

C	P(X=0)	P(X=1)
0	4/7	3/7
1	2/7	5/7

C	X	P(Y=0)	P(Y=1)
0	0	3/4	1/4
0	1	2/3	1/3
1	0	1/2	1/2
1	1	1/5	4/5

Introdução (7/9)

- **Mais exemplos...**

graph	vertices	edges
communication	telephones, computers	fiber optic cables
circuits	gates, registers, processors	wires
mechanical	joints	rods, beams, springs
hydraulic	reservoirs, pumping stations	pipelines
financial	stocks, currency	transactions
transportation	street intersections, airports	highways, airway routes
scheduling	tasks	precedence constraints
software systems	functions	function calls
internet	web pages	hyperlinks
games	board positions	legal moves
social relationship	people, actors	friendships, movie casts
neural networks	neurons	synapses
protein networks	proteins	protein-protein interactions
chemical compounds	molecules	bonds

https://www.cs.princeton.edu/~rs/AlgsDS07/11UndirectedGraphs.pdf

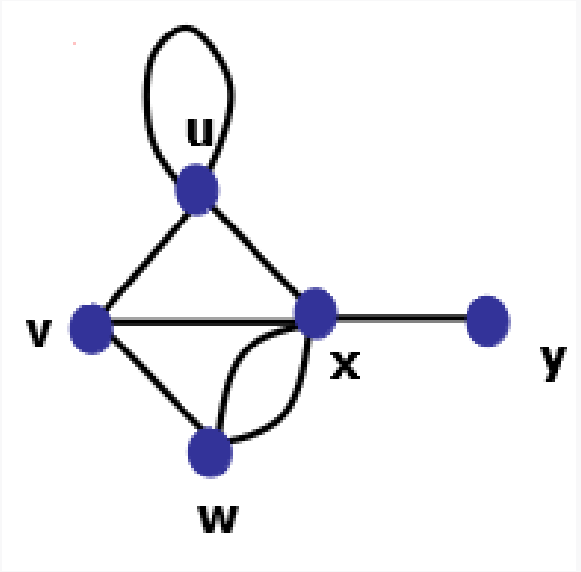
Introdução (8/9)

- **Grafo - Definição Formal**
  - Um grafo **G** é representado por:

    $G=(V(G), E(G), \psi_G)$

    - $V(G)$: Conjunto não vazio de vértices.
    - $E(G)$: Conjunto disjunto de V(G), chamado arestas.
    - ψ_G : Função que associa cada aresta de G um par de vértices de G

G

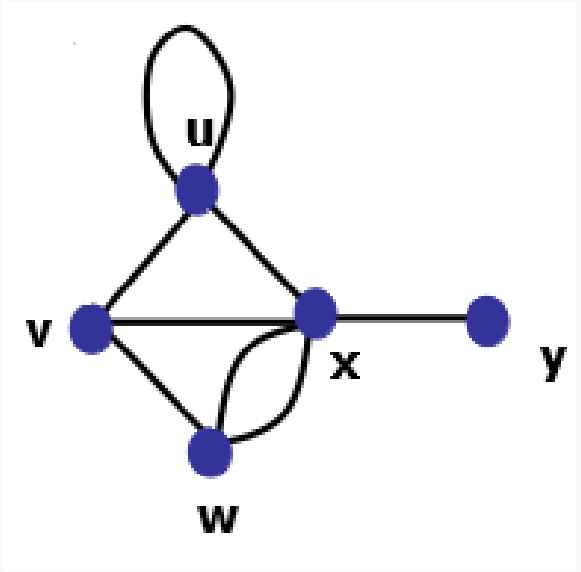


* Definição retirada de Nogueira (2019).

Introdução (9/9)

- **Grafo – Definição Formal**
  - Exemplo:
    - V(G) ={u, v, w, x, y}
    - E(G)={a, b, c, d, e, f, g, h}
    - ψ_G(a)= (u, v),
    - ψ_G (b)= (u, u),
    - ψ_G(c)= (v, w),
    - ψ_G(d)= (w, x),
    - ψ_G(e)= (v, x),
    - ψ_G(f)= (w, x),
    - ψ_G(g)= (u, x),
    - ψ_G(h)= (x, y)

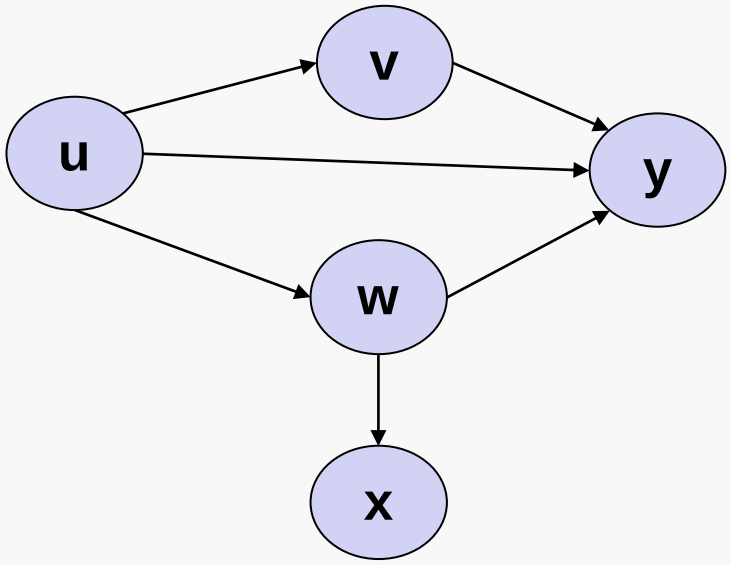
G



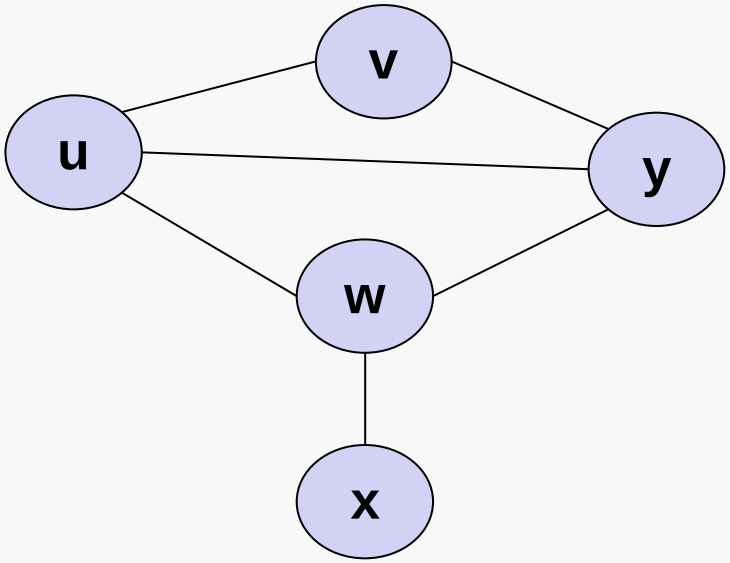
* **Fonte:** Nogueira (2019).

Tipos de Grafos (1/4)

- **Grafos Orientados**
  - As arestas possuem uma **direção**.
    - Também são chamados de grafos dirigidos ou **dígrafos**.

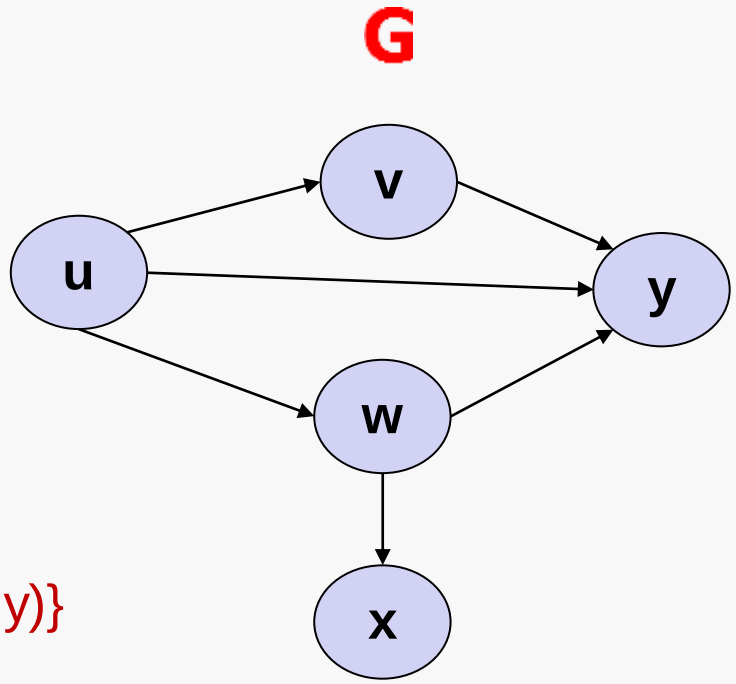


- **Grafos Não Orientados**
  - As arestas possuem são **bi-direcionais**.
    - Se existe uma conexão $u \rightarrow v$, então também existe a conexão $v \rightarrow u$



Tipos de Grafos (2/4)

- **Grafo Orientado – Exemplo**

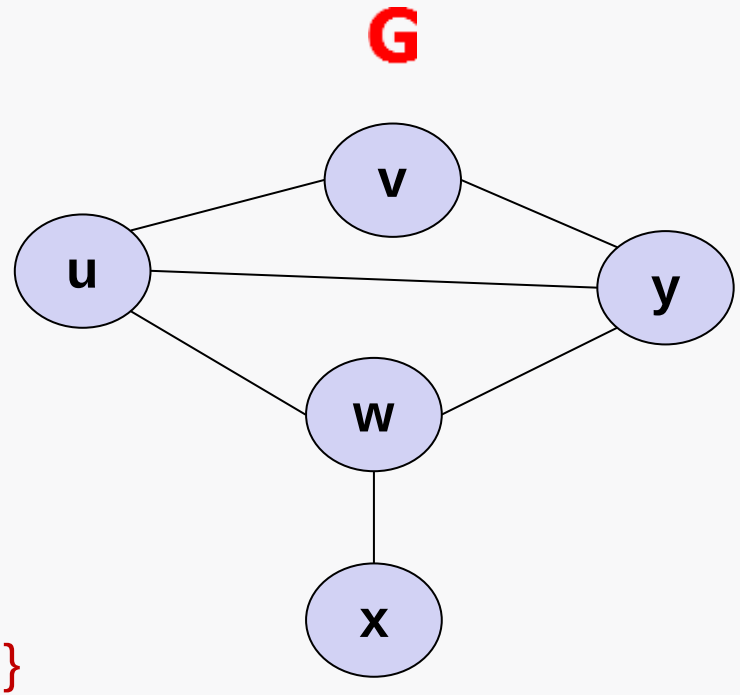


  - G = (V,E)
  - V(G) = {u, v, w, x, y}
  - E(G) = {(u,v), (u,w), (u,y), (v,y), (w,x), (w,y)}

- **Exemplos de Aplicações**
  - Redes sociais:
    - Pessoa “u” segue a “w”, mas nem sempre o oposto é verdadeiro
  - Internet:
    - Página “u” tem um link para a “w”, mas o contrário pode não ocorrer.

Tipos de Grafos (3/4)

- **Grafo Não Orientado – Exemplo**

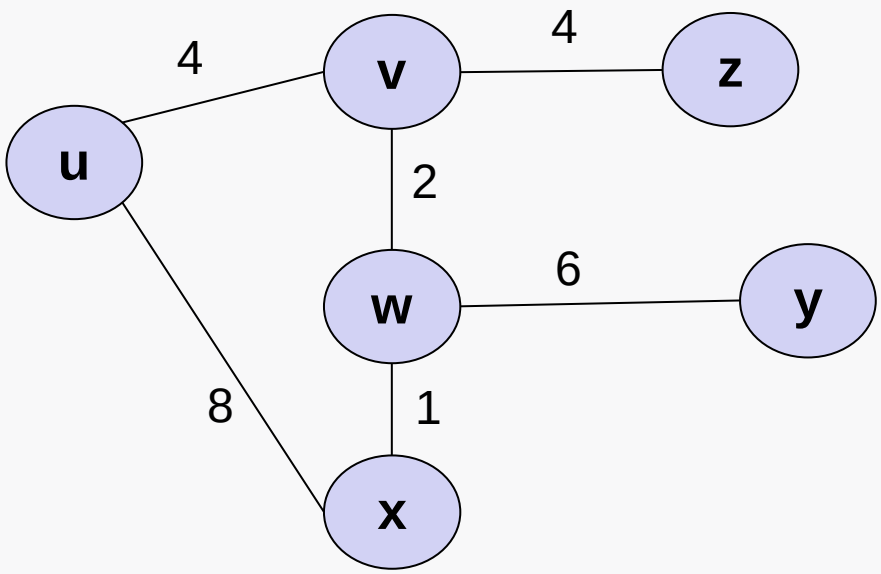


  - G = (V,E)
  - V(G) = {u, v, w, x, y}
  - E(G) = {(u,v), (u,w), (u,y), (v,u), (v,y), (w,u), (w,x), (w,y), (x,w), (y,u), (y,v), (y,w)}

- **Exemplos de Aplicações**
  - Dicionários
    - Se palavra “u” é um sinônimo de “w”, então “w” é sinônimo de “u”.
  - Mapas
    - Se país “u” faz fronteira com “w”, então “w” faz fronteira com “u”

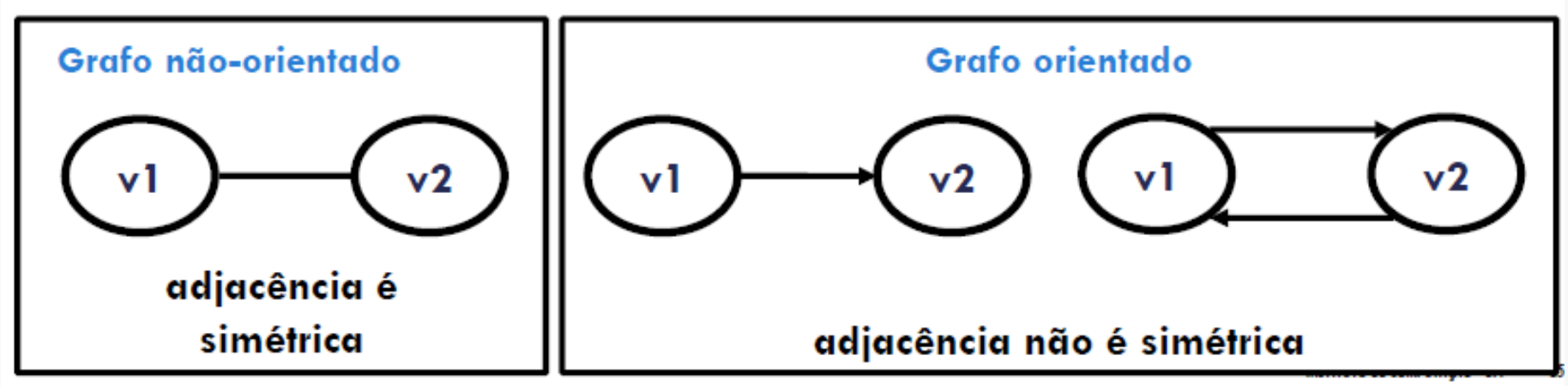
Tipos de Grafos (4/4)

- **Grafo Valorado**
  - Possui valores (pesos) associados às arestas e/ou vértices.
  - Os pesos podem representar:
    - **custo** ou **distância**:
      - Qual a quantidade de esforço necessária para chegar de um nó a outro?
    - **capacidade**:
      - Qual a quantidade máxima de fluxo que pode ser transportada de um vértice a outro?
      - Qual o valor máximo suportado pro um vértice?



Terminologia (1/5)

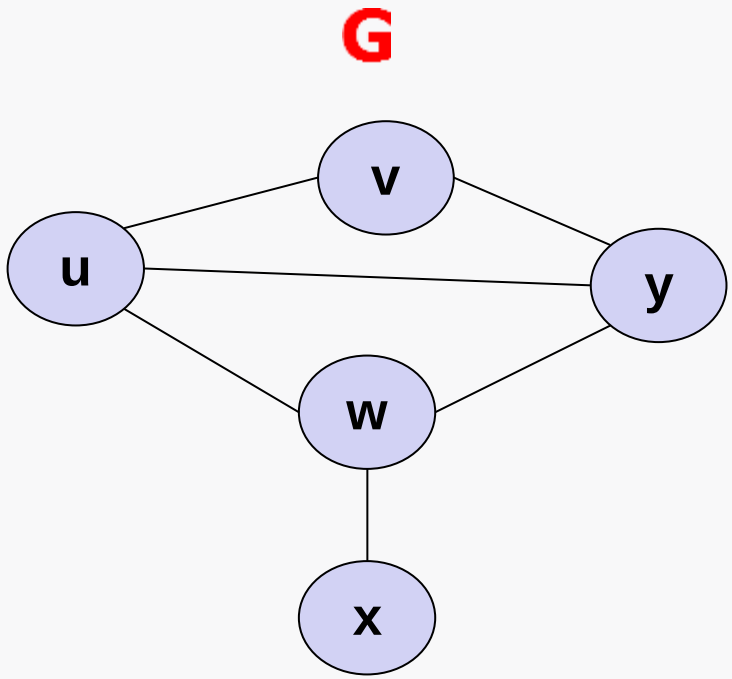
- **Vértices Adjacentes**
  - Um vértice v1 é **adjacente** a um vértice v2, se existe uma aresta conectando v1 a v2.
    - Em grafo **não orientado**: v1 é adjacente a v2 se existe aresta {v1,v2}. Nesse caso v2 também é adjacente a v1.
    - Em **grafo orientado**: v1 é adjacente a v2 se existe aresta (v1, v2)



* Figura retirada de https://sites.google.com/ic.uff.br/estrutura-de-dados-icuff/home

Terminologia (2/5)

- **Grau de um Vértice**
  - Corresponde ao número de arestas que nele ou dele incidem.
    - Graus dos nós do exemplo ao lado:
      - u: 3
      - v: 2
      - w: 3
      - y: 3
      - x: 1



- **Ordem do Grafo**
  - Número de vértices que ele possui.
    - Ordem(G) = |V|

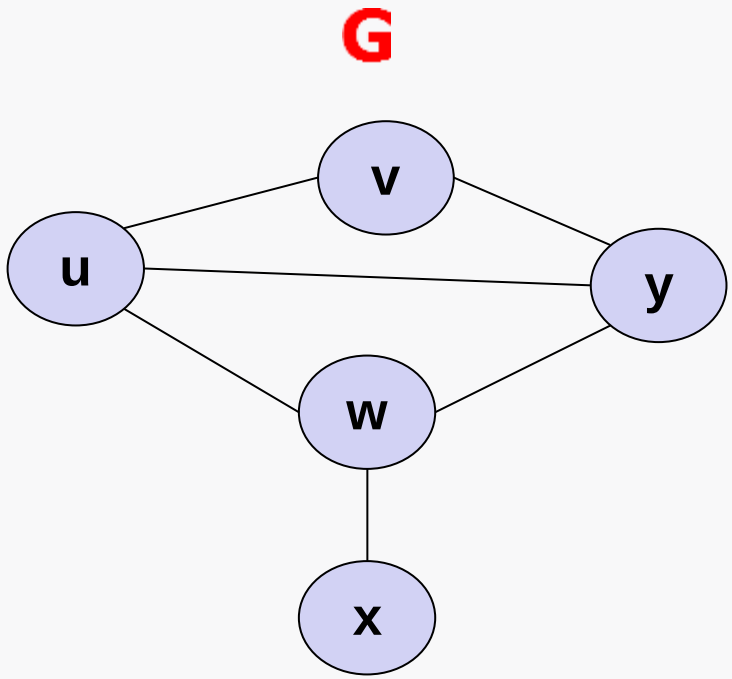
Grafo de ordem 5

Terminologia (3/5)

- **Caminho**
  - Sequência de vértices e arestas para chegar de um nó v1 a um nó v2

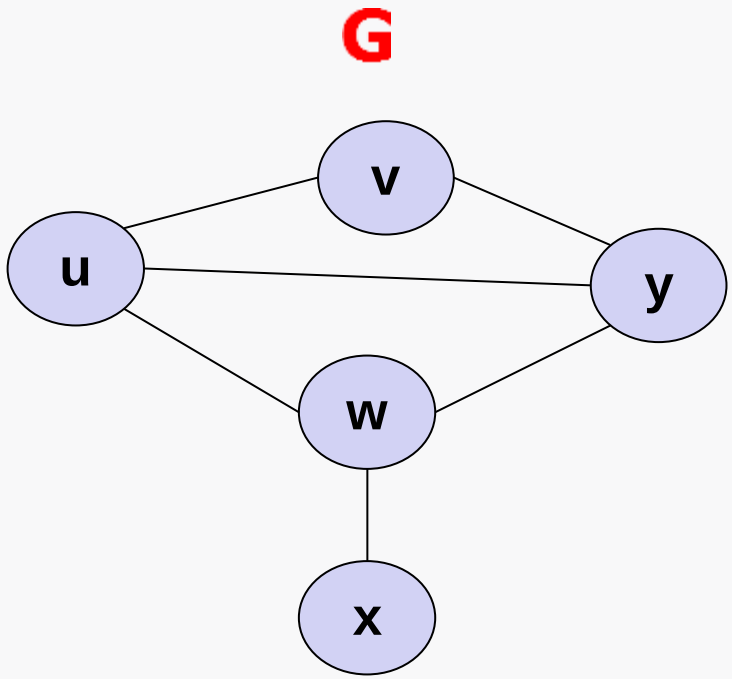
  - Ex1: Caminho do nó “u” ao “v”
    - Tamanho = 1
    - Vértices = {u, v}
    - Arestas = {(u, v)}

  - Ex1: Caminho do nó “v” ao “x”
    - Tamanho = 3
    - Vértices = {v, y, w, x}
    - Arestas = {(v, y), (w,y), (w,x)}



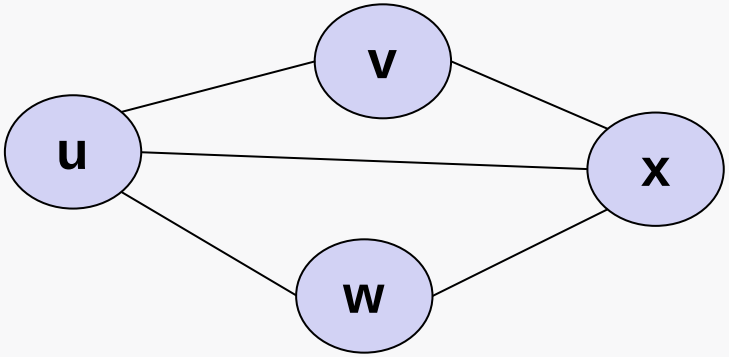
Terminologia (4/5)

- **Ciclos**
  - Um caminho $<V_1, V_2, ..., V_k>$ forma um **ciclo** se $V_1 = V_k$
  - Por exemplo, no grafo ao lado é possível fazer um caminho do nó u até o nó u, passando por v e y.
    - Esse caminho é um ciclo.



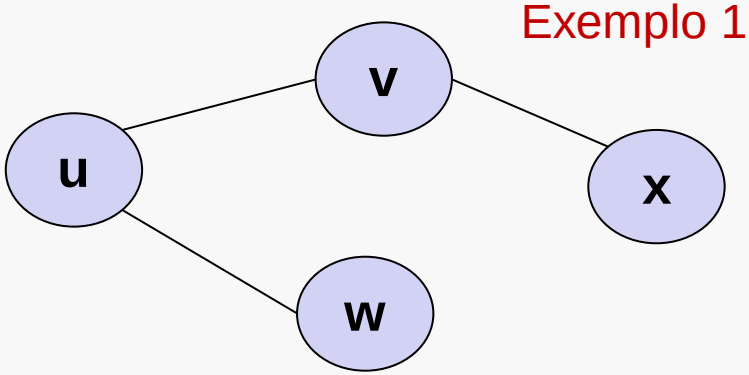
Terminologia (5/5)

- **Grafo Cíclico**
  - Possui ciclos

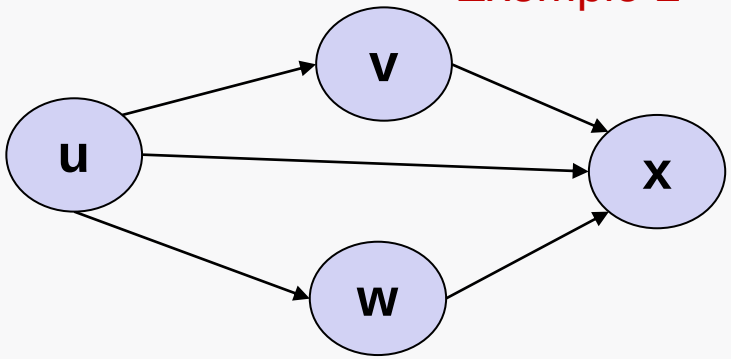


- **Grafos Acíclico**
  - Sem ciclos

Exemplo 1



Exemplo 2



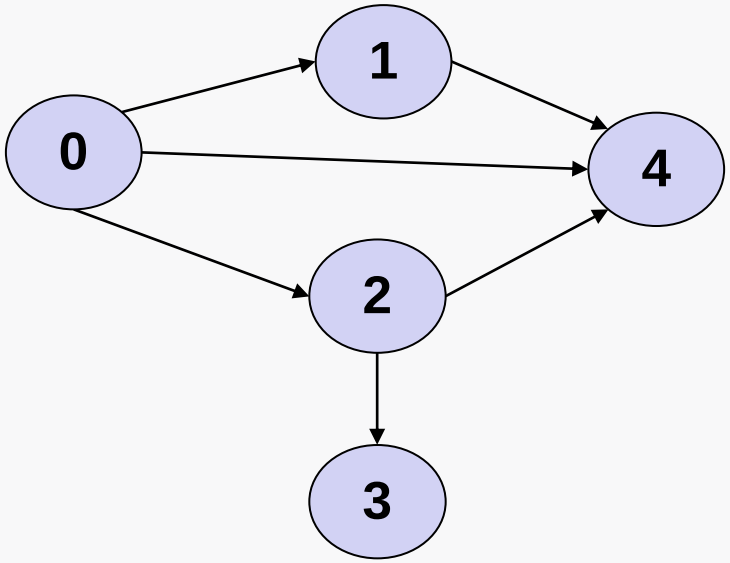
Criação de Grafos (1/9)

- **Representando um grafo:**
  - Há algumas formas diferentes de representar grafos.
  - Nesta aula, veremos as duas mais comuns:
    1. Matriz de Adjacência
    2. Lista de Adjacência
  - Dependendo da aplicação e do tipo do grafo, qualquer uma das representações poderá oferecer vantagens e desvantagens.
  - O Python possui alguns pacotes para grafos:
    - https://wiki.python.org/moin/PythonGraphLibraries
    - Porém nesta aula vamos criar nossos grafos “na mão”.

Criação de Grafos (2/9)

- **Matriz de Adjacência:**
  - Matriz *n x n*, onde *n* é o número de vértices do grafo.
  - Cada elemento $a_{i,j}$ dessa matriz é:
    - 1 se existe uma aresta de v_i para v_j.
    - 0 se não existe.

  - **Exemplo 1**: grafo direcionado



	0	1	2	3	4
0	0	1	1	0	1
1	0	0	0	0	1
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	0	0	0

Criação de Grafos (3/9)

- **Matriz de Adjacência:**
  - Em Python, podemos utilizar: listas 2d, arrays NumPy ou Dataframes pandas para implementar a matriz de adjacência.

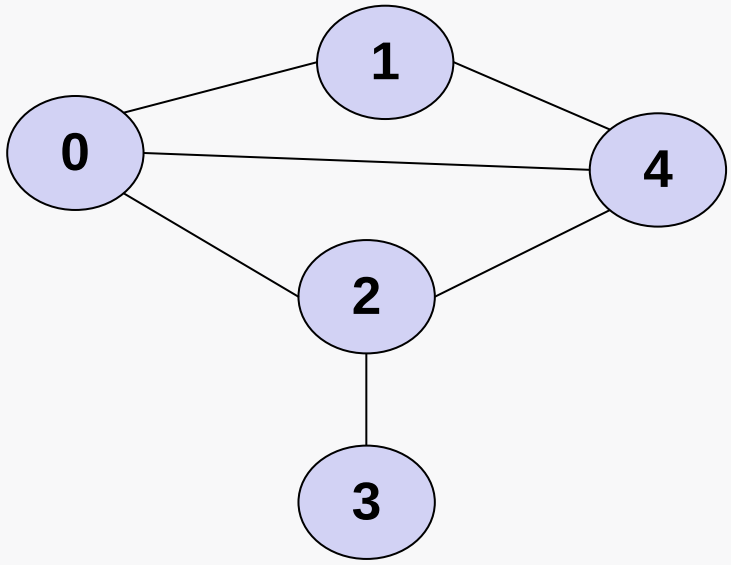
```
import numpy as np

n = 5
g = np.zeros((n,n), dtype="int")

g[0,[1,2,4]]=1
g[1,4]=1
g[2,[3,4]]=1

print(g)
```

	0	1	2	3	4
0	0	1	1	0	1
1	0	0	0	0	1
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	0	0	0

```
>
[[0 1 1 0 1]
 [0 0 0 0 1]
 [0 0 0 1 1]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

Criação de Grafos (4/9)

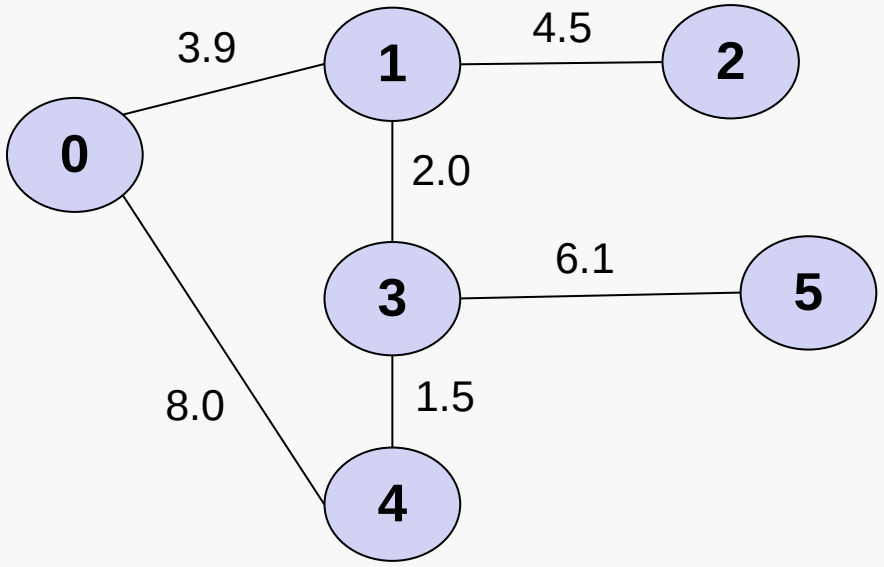
- **Matriz de Adjacência**
  - Exemplo 2 – **grafo não direcionado**
    - Neste caso a matriz será **simétrica**
    - A parte em azul pode ser preenchida ou não, dependendo da forma com que você utilizará o programa.



	0	1	2	3	4
0	0	1	1	0	1
1	1	0	0	0	1
2	1	0	0	1	1
3	0	0	1	0	0
4	1	1	1	0	0

Criação de Grafos (5/9)

- **Matriz de Adjacência**
  - Exemplo 3 – **grafo valorado**
    - Basta armazenar os pesos nas células
    - Em geral, precisamos “marcar” os pares de arestas não conectadas (usando np.nan ou np.inf, por exemplo)



	0	1	2	3	4	5
0	-	3.9	-	-	8.0	-
1	3.9	-	4.5	2.0	-	-
2	-	4.5	-	-	-	-
3	-	3.1	-	-	1.5	6.1
4	8.0	-	-	1.5	-	-
5	-	-	-	6.1	-	-

Criação de Grafos (6/9)

- **Matriz de Adjacência:**
  - Implementação na NumPy

	0	1	2	3	4	5
0	-	3.9	-	-	8.0	-
1	3.9	-	4.5	2.0	-	-
2	-	4.5	-	-	-	-
3	-	3.1	-	-	1.5	6.1
4	8.0	-	-	1.5	-	-
5	-	-	-	6.1	-	-

```
from  pprint import pprint
import numpy as np
n = 6

g = np.repeat(np.nan, n*n)
g = g.reshape(n,n)

g[[0,1],[1,0]]=3.9
g[[0,4],[4,0]]=8.0
g[[1,2],[2,1]]=4.5
g[[1,3],[3,1]]=2.0
g[[3,4],[4,3]]=1.5
g[[3,5],[5,3]]=6.1

print(g)
```

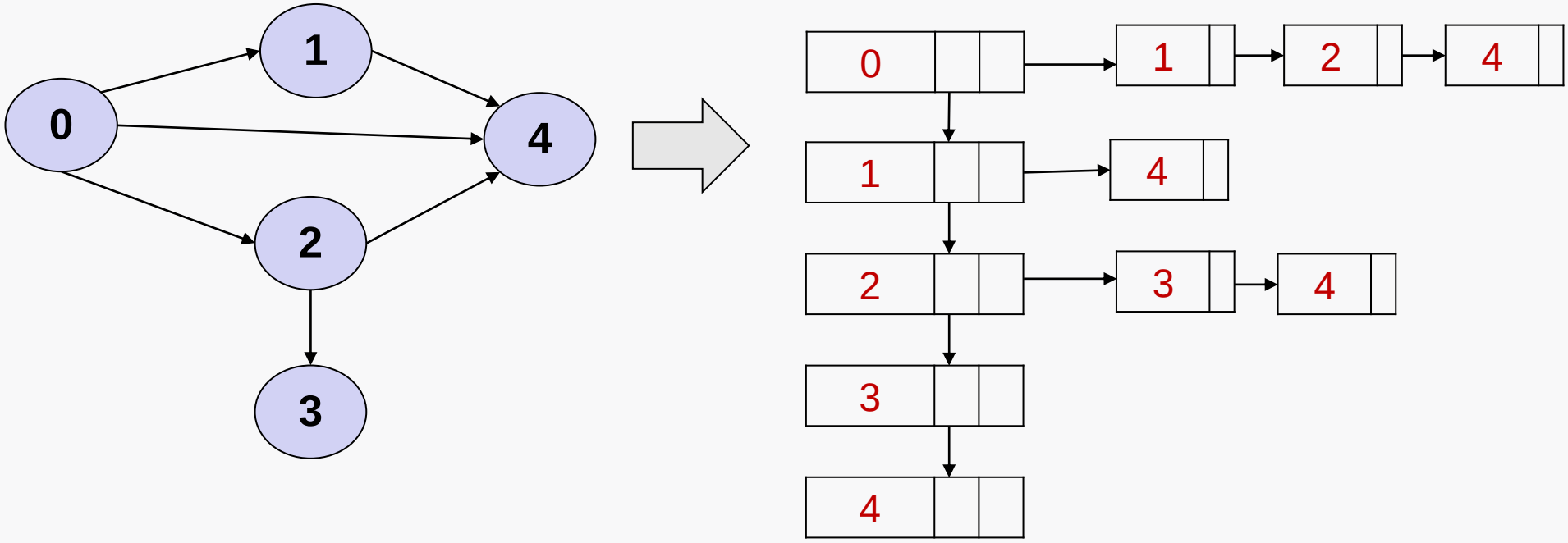
```
>
array([[nan, 3.9, nan, nan, 8. , nan],
       [3.9, nan, 4.5, 2. , nan, nan],
       [nan, 4.5, nan, nan, nan, nan],
       [nan, 2. , nan, nan, 1.5, 6.1],
       [8. , nan, nan, 1.5, nan, nan],
       [nan, nan, nan, 6.1, nan, nan]])
```

Criação de Grafos (7/9)

- **Matriz de Adjacência:**
  - **Vantagens**:
    - Simples
    - Acesso por índice a qualquer elemento $a_{i,j}$ em tempo O(1)
    - Fácil determinar se uma aresta existe no grafo.
    - Fácil determinar os vértices adjacentes a qualquer vértice v
    - Fácil inserir nova aresta
  - **Desvantagens**:
    - Se *n* é muito grande, ocupa muito espaço em memória.
    - Inserir novo vértice é trabalhoso.

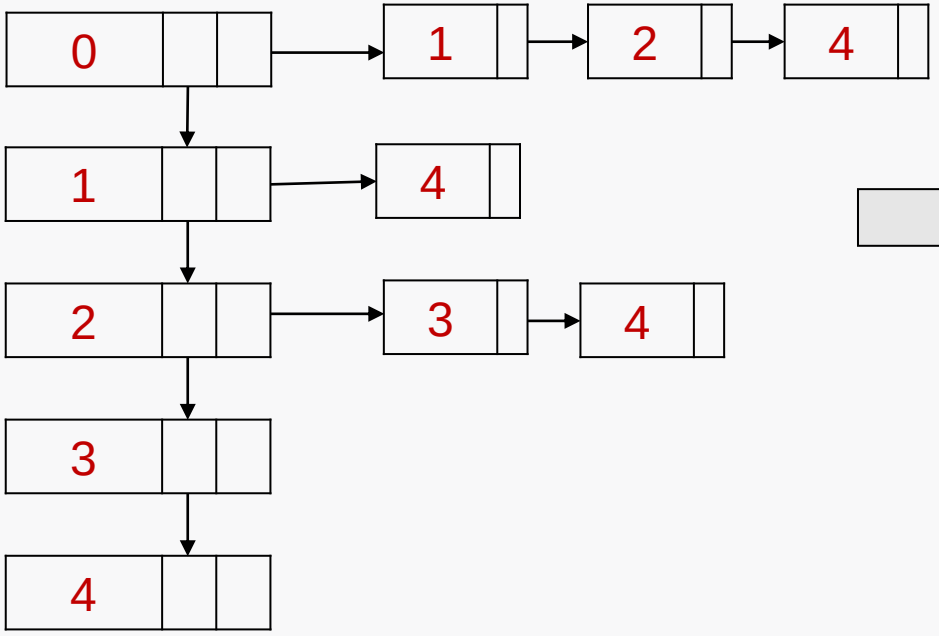
Criação de Grafos (8/9)

- **Lista de Adjacência – outra forma de representar o grafo**
  - Para cada vértice v, é representada a lista de vértices u, de modo que (v, u) ∈ G.
  - **Vantagens**: economia de memória e flexibilidade.
  - **Desvantagem**: não tão simples quanto a matriz de adjacência.



Criação de Grafos (9/9)

- **Lista de Adjacência**
  - A maneira clássica de implementar lista de adjacências é usando duas listas encadeadas ou um vetor de listas encadeadas
  - Em Python, uma forma mais simples e muitas vezes possível, consiste no uso de um **dicionário de listas***.




```
g = {   "0" : ["1", "2", "4"],
        "1" : ["4"],
        "2" : ["3", "4"],
        "3" : [],
        "4" : []
    }
```

* *Essa forma de implementação é comumente referenciada como* ***mapa de adjacências***

Operações (1/8)

- Nos próximos slides implementaremos um grafo através de um **mapa de adjacências** e **POO**.

- As seguintes operações serão definidas:

Operação	Descrição
Criação	Cria um grafo vazio ou a partir de uma sequência de vértices.
get_vertices	Retorna todos os vértices do grafo
get_arestas	Retorna todas as arestas do grafo
inserir_vertice	Insere um vértice, com sua lista de arestas vazia.
inserir_aresta	Insere uma aresta entre dois vértices
__str__	Gera uma string com a representação do grafo.

- O modelo foi baseado no exemplo apresentado em: https://www.python-course.eu/graphs_python.php

Operações (2/8)

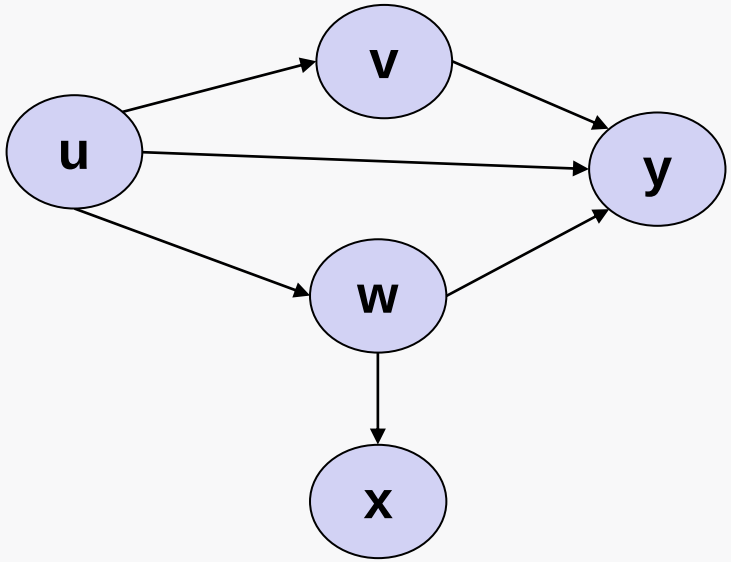
- **Criação + inserir vertices**

```
# implementação de um grafo usando lista de adjacências + POO

class Grafo():
    #cria um grafo vazio
    def __init__(self):
        self._g = {}

 #insere um vértice, caso o mesmo não exista no grafo
    def inserir_vertice(self, v):
        if v not in self._g:
            self._g[v] = []
```

Operações (3/8)

- Vamos testar usando o grafo ao lado.

```
if __name__ == "__main__":

    G = Grafo()

    G.inserir_vertice("u")
    G.inserir_vertice("v")
    G.inserir_vertice("w")
    G.inserir_vertice("x")
    G.inserir_vertice("y")

    print(G._g)

>
{'u': [], 'v': [], 'w': [], 'x': [], 'y': []}
```



Operações (4/8)

- **Inserir arestas**

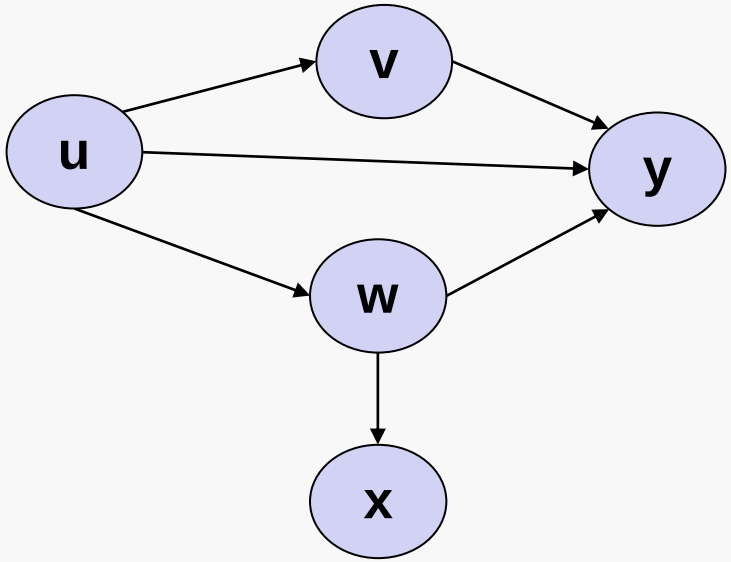
```
# insere aresta de v1 para v2
# - a entrada deve ser uma tupla (v1,v2)
def inserir_aresta(self, par_vertices):
    v1 = par_vertices[0]
    v2 = par_vertices[1]

    #se v1 não existe, cancela a operação
    if v1 not in self._g:
        return False
    #se v1 existe, prossegue...
    else:
        if v2 not in self._g[v1]: #aresta v1->v2 não existe, vou inserir
            self._g[v1].append(v2)
            return True
        else: #aresta v1->v2 já existe, operação cancelada
            return False
```

Operações (5/8)

- Continuando nosso teste.

```
if __name__ == "__main__":

    …
    G.inserir_aresta(("u","v"))
    G.inserir_aresta(("u","w"))
    G.inserir_aresta(("u","y"))
    G.inserir_aresta(("w","x"))
    G.inserir_aresta(("w","y"))
    G.inserir_aresta(("v","y"))

    print(G._g)
```

```
>
{'u': ['v', 'w', 'y'], 'v': ['y'], 'w': ['x', 'y'],
'x': [], 'y': []}
```



Operações (6/8)

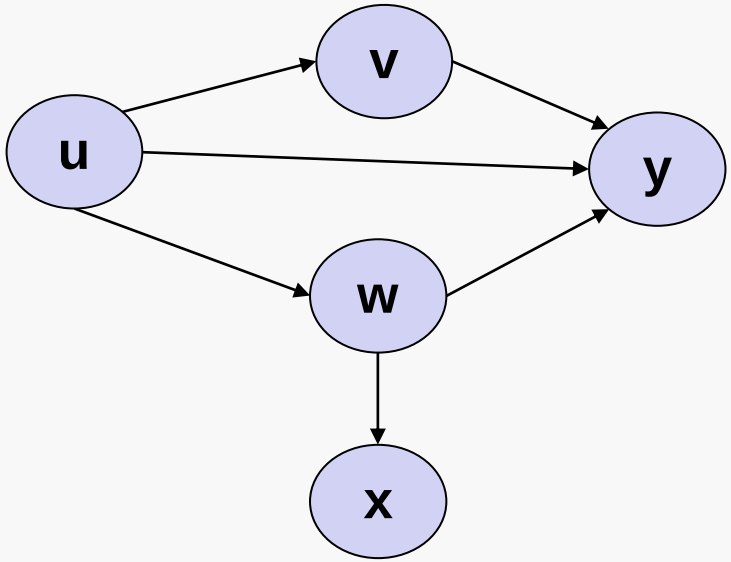
- **get_vertices e get_arestas**

```
#retorna os vértices do grafo em um conjunto
def get_vertices(self):
    return set(self._g.keys())

#retorna as arestas do grafo em um conjunto de tuplas
def get_arestas(self):
    arestas = set()
    for v in self._g:
        for vizinho in self._g[v]:
            arestas.add((v, vizinho))

    return arestas
```

Operações (7/8)

- Continuando nosso teste.

```
if __name__ == "__main__":

    …
    V = G.get_vertices()
    print("V = ", V)

    E = G.get_arestas()
    print("E = ", E)
```




```
>
V =  {'x', 'u', 'w', 'v', 'y'}

E =  {('u', 'w'), ('w', 'y'), ('u', 'v'), ('v', 'y'), ('u', 'y'), ('w', 'x')}
```

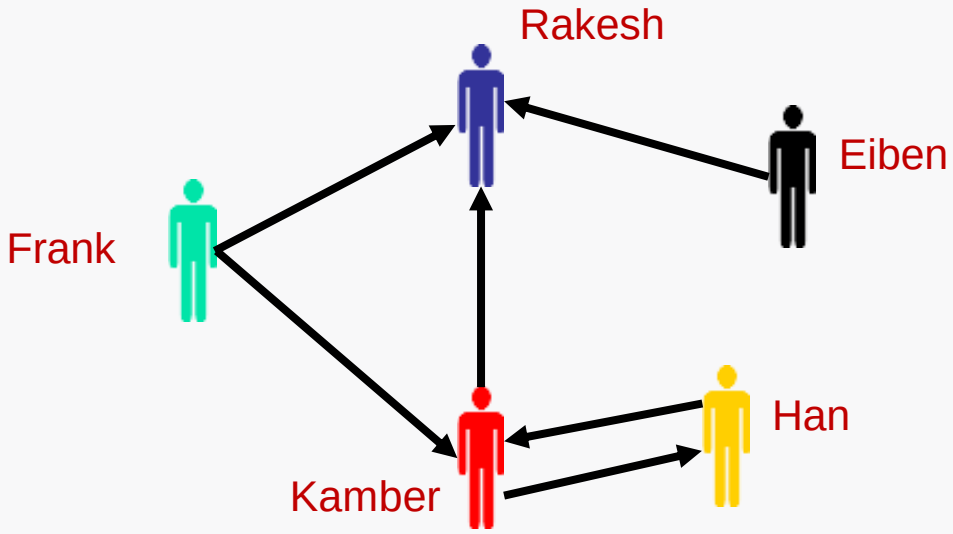
Operações (8/8)

- **Representação string do grafo**

```
# representação string do grafo
def __str__(self):
        aux = "V = " + str(self.get_vertices())
        aux += "\nE = " + str(self.get_arestas())
        return aux



if __name__ == "__main__":

    …
    print(G)

>
V = {'w', 'v', 'u', 'y', 'x'}
E = {('w', 'y'), ('u', 'w'), ('u', 'y'), ('w', 'x'), ('u', 'v'), ('v', 'y')}
```

Exercícios

1. Faça um programa que importe a classe Grafo para representar a rede abaixo:



2. Modificar a classe Grafo

- Modifique a classe Grafo para que ela também ofereça suporte à exclusão de arestas e vértices. Considere que os grafos sejam direcionados.
  - **remover_aresta(par_vertices)**: remove aresta de v1 para v2. A entrada deve ser uma tupla (v1, v2).
  - **remover_vertice(v)**: remove o vértice v, todas as arestas que partem dele e todas que nele chegam (DICA: utilize remover_aresta() dentro desse método)

Referências

- Python 3 Tutorial – Object Oriented Programming. https://www.python-course.eu/graphs_python.php

- Nogueira, L. T. (2019). Teoria dos Grafos. http://www.ic.uff.br/~loana/teaching/teoria-dos-grafos--loana/