

Algoritmos em Grafos

Praticas de implementação

Objetivo

- Apresentar os principais recursos da linguagem Python e do ambiente de desenvolvimento Python(x, y) no contexto de aplicações práticas de algoritmos em grafos.

Por que Python?

- Sintaxe permite produzir código mais conciso e legível (próximo a notação matemática).
- Interpretador interativo que permite testar o código de modo rápido e fácil.
- Alto poder de expressão e flexibilidade.
- Portabilidade.
- Variedade muito grande de bibliotecas.
- Software livre.

Python(x, y)/Spyder

- Une todas as ferramentas necessárias:
 - Estrutura de projetos.
 - Editor de texto.
 - Inspetor de objetos.
 - Inspetor de variáveis.
 - File Explorer.
 - Console Python (Interpretador/Ipython).

Python(x, y)/Spyder

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `teste.py` with the following code:

```
1 # -*- coding: utf-8 -*-
2
3 def fib(n):
4     """
5     Descrição
6     -----
7     Algoritmo iterativo para a sequência de Fibonacci.
8
9     Entradas
10    -----
11    n : um número inteiro positivo.
12    """
13    i = 1
14    j = 0
15
16    for k in range(n):
17        t = i + j
18        i = j
19        j = t
20
21    return j
22
23
24
```

The right-hand side of the interface features the Object inspector, which displays the `fib` function. It includes the following information:

- Definition:** `fib(n)`
- Type:** Function of teste module
- Descrição:** Algoritmo iterativo para a sequência de Fibonacci.
- Entradas:** n : um número inteiro positivo.

Below the Object inspector is the IPython console, which shows the execution of the `fib` function for various inputs:

```
In [4]: fib(0)
Out[4]: 0

In [5]: fib(1)
Out[5]: 1

In [6]: fib(2)
Out[6]: 1

In [7]: fib(3)
Out[7]: 2

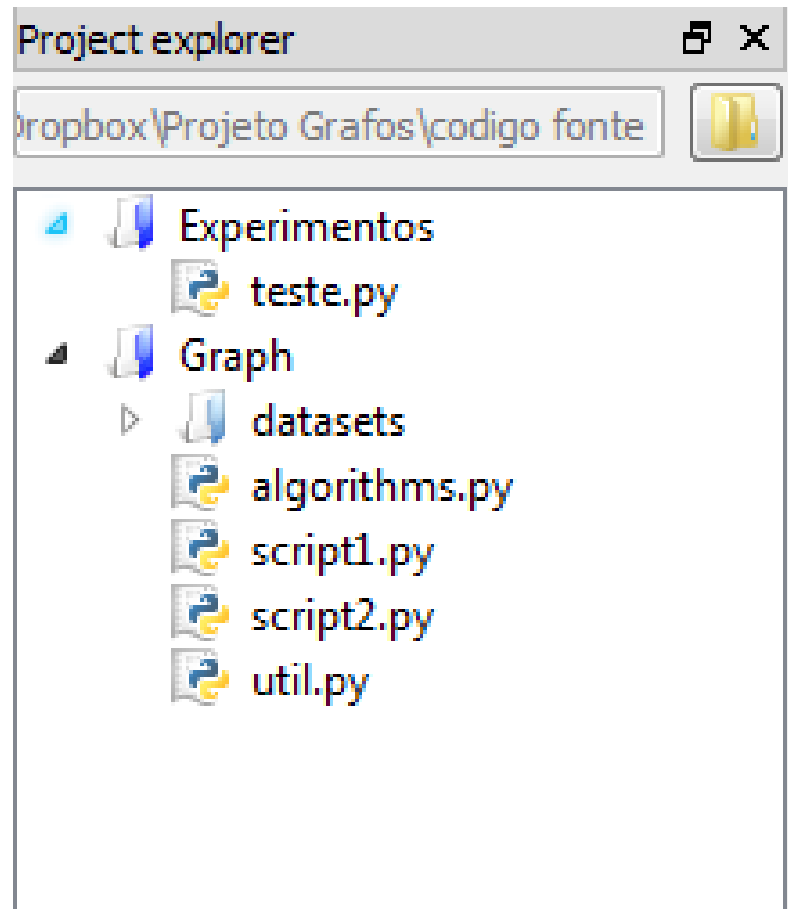
In [8]:
```

The bottom status bar indicates the following details:

- Permissions: RW
- End-of-lines: CRLF
- Encoding: UTF-8
- Line: 24
- Column: 5
- Memory: 45 %

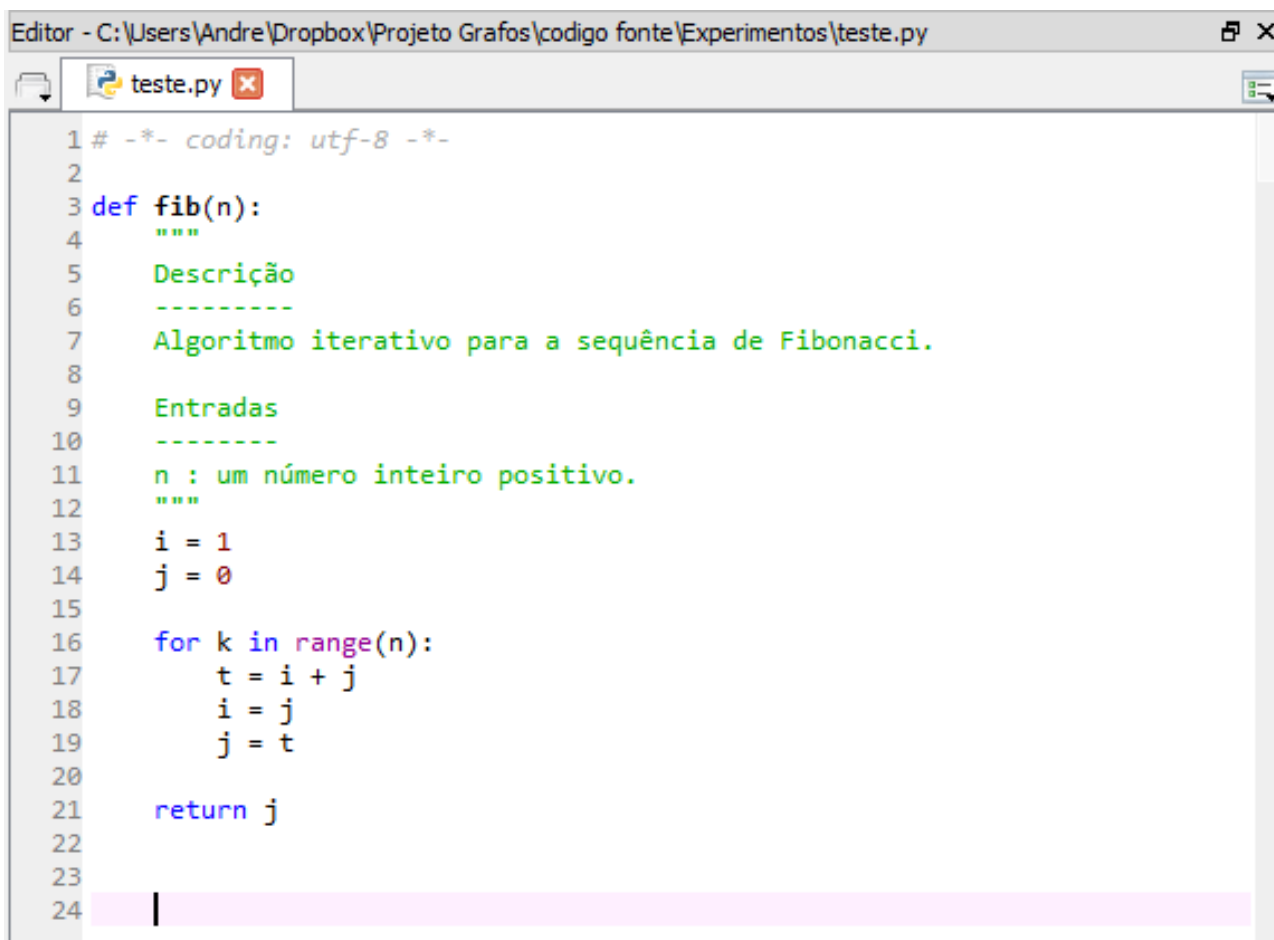
Python(x, y)/Spyder

- Estrutura de projetos:



Python(x, y)/Spyder

- Editor de texto:

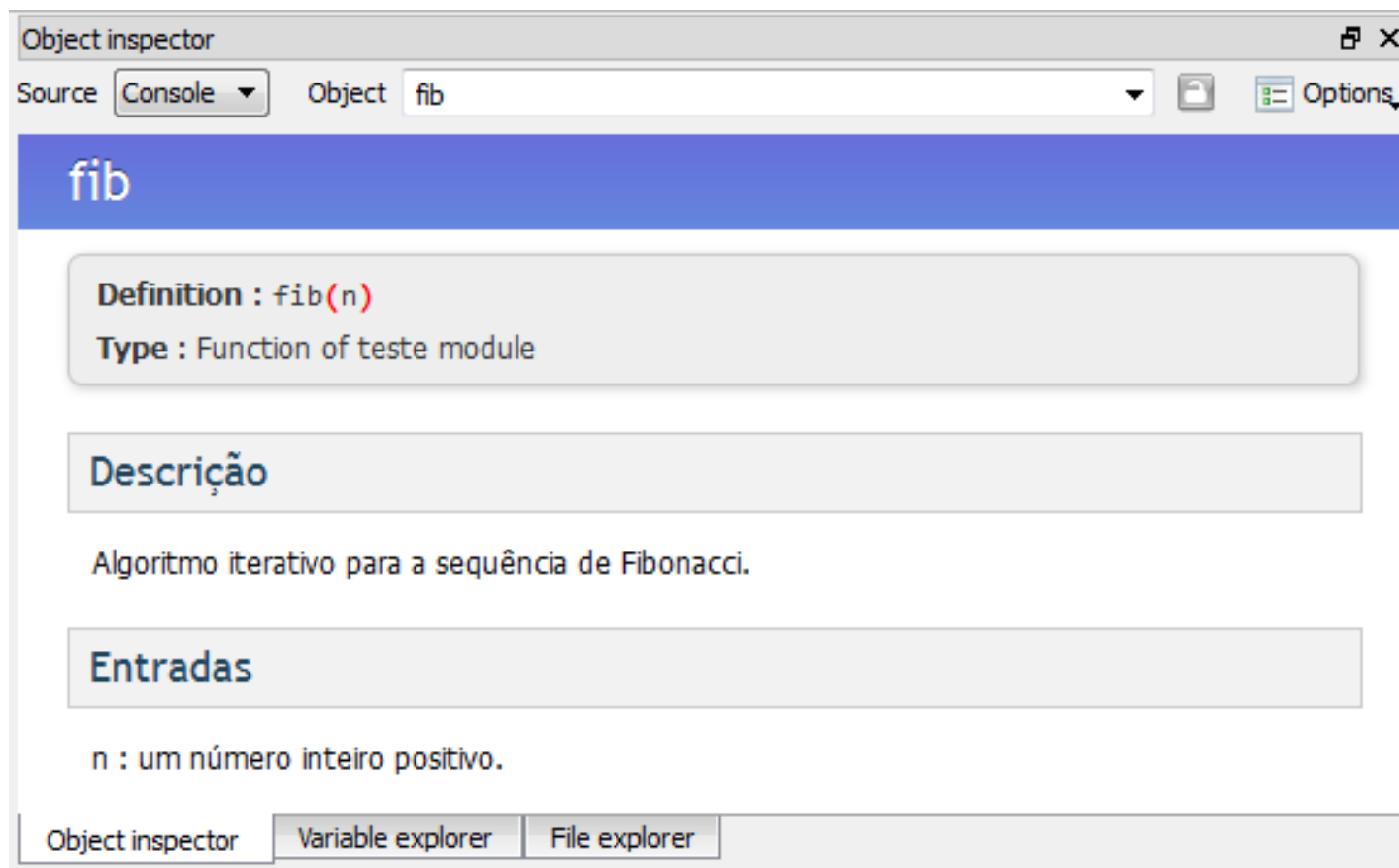


The image shows a screenshot of a text editor window titled "Editor - C:\Users\Andre\Dropbox\Projeto Grafos\codigo fonte\Experimentos\teste.py". The editor contains a Python script for calculating the nth Fibonacci number. The code is as follows:

```
1 # -*- coding: utf-8 -*-
2
3 def fib(n):
4     """
5     Descrição
6     -----
7     Algoritmo iterativo para a sequência de Fibonacci.
8
9     Entradas
10    -----
11    n : um número inteiro positivo.
12    """
13    i = 1
14    j = 0
15
16    for k in range(n):
17        t = i + j
18        i = j
19        j = t
20
21    return j
22
23
24
```

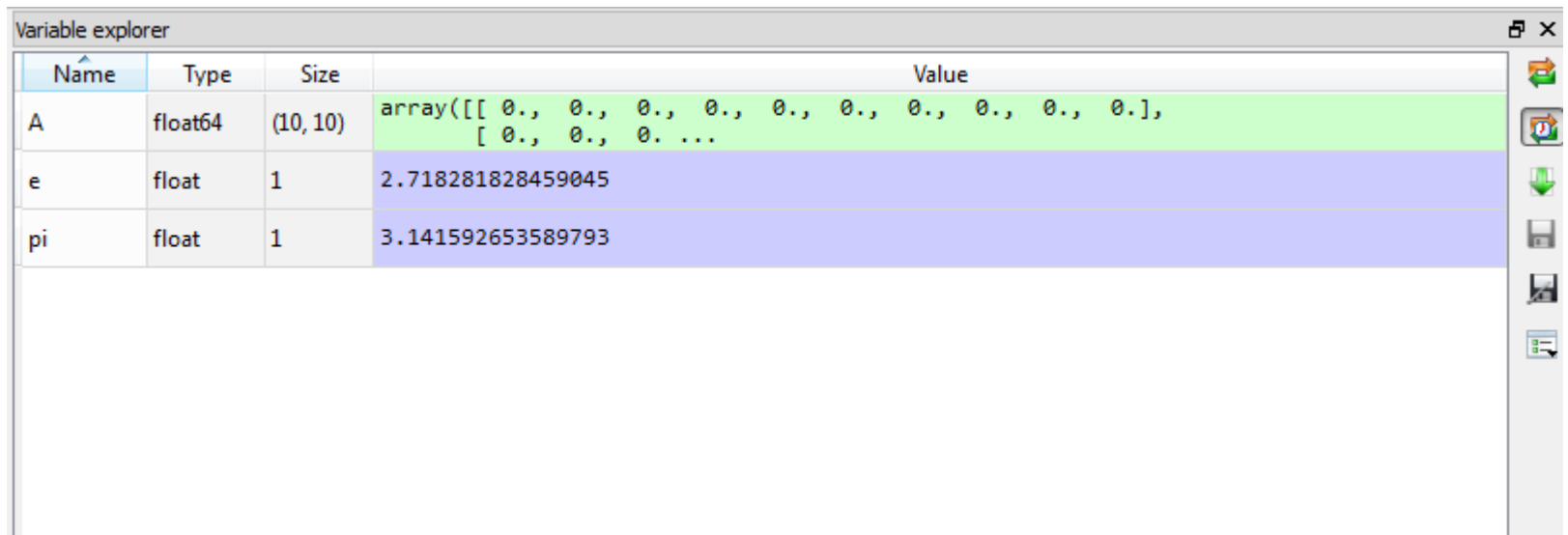
Python(x, y)/Spyder

- Inspetor de objetos:



Python(x, y)/Spyder

- Inspetor de variáveis.

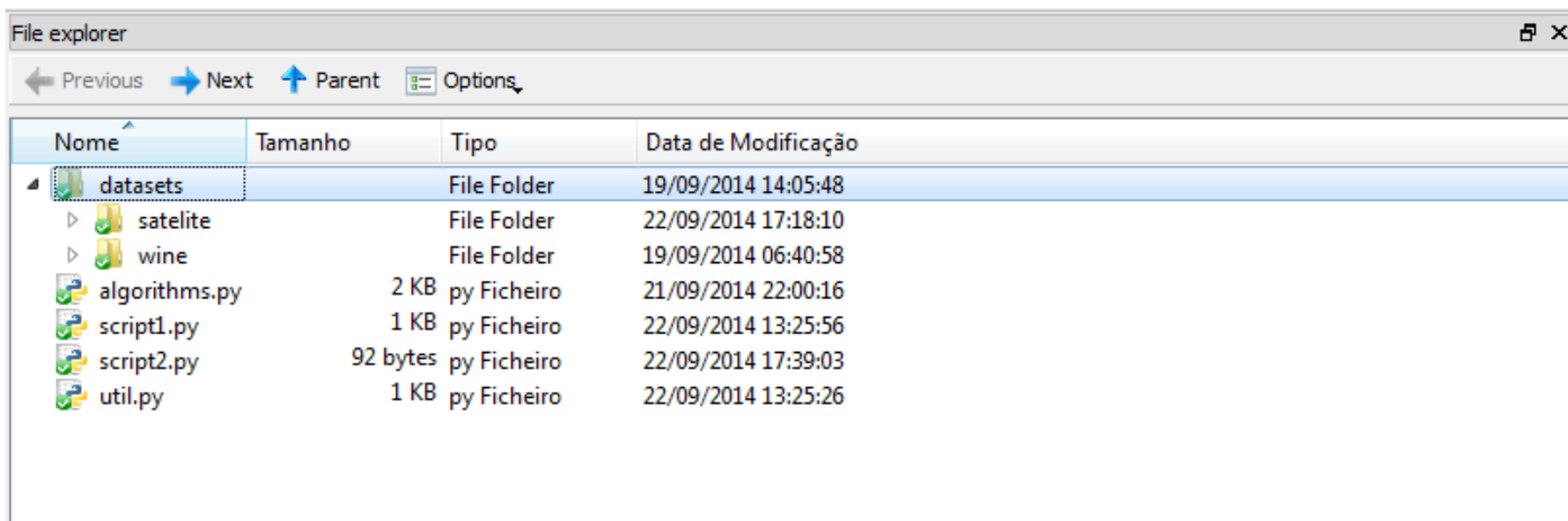


The image shows the 'Variable explorer' window from the Spyder Python IDE. It contains a table with three columns: 'Name', 'Type', 'Size', and 'Value'. The first row, for variable 'A', has a green background. The second and third rows, for variables 'e' and 'pi', have a blue background. On the right side of the window, there is a vertical toolbar with icons for adding, removing, and refreshing variables.

Name	Type	Size	Value
A	float64	(10,10)	array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], [0., 0., 0. ...
e	float	1	2.718281828459045
pi	float	1	3.141592653589793

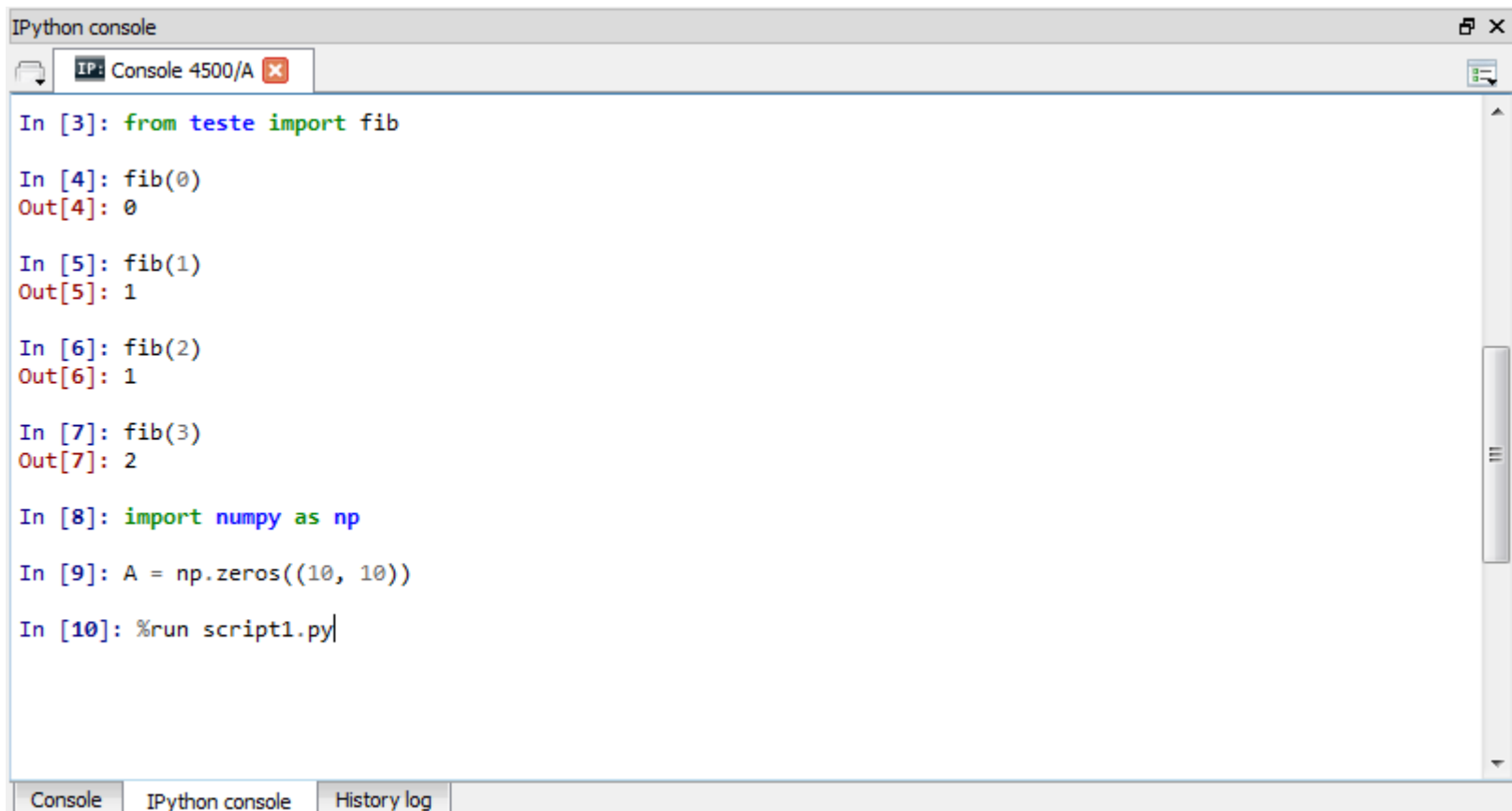
Python(x, y)/Spyder

- File Explorer:



Python(x, y)/Spyder

- Console Python (Interpretador/Ipython).



The screenshot shows the IPython console window within the Spyder IDE. The window title is "IPython console". The console area displays a series of Python commands and their outputs. The commands include importing a module, calling a function, importing numpy, and running a script. The outputs show the results of these commands, such as the values of the Fibonacci sequence and the creation of a numpy array.

```
IPython console
IP: Console 4500/A

In [3]: from teste import fib
In [4]: fib(0)
Out[4]: 0
In [5]: fib(1)
Out[5]: 1
In [6]: fib(2)
Out[6]: 1
In [7]: fib(3)
Out[7]: 2
In [8]: import numpy as np
In [9]: A = np.zeros((10, 10))
In [10]: %run script1.py
```

At the bottom of the window, there are three tabs: "Console", "IPython console", and "History log". The "IPython console" tab is currently selected.

Linguagem Python - Tipos

- Booleano:
 - Literais **True** e **False**.
 - Operadores lógicos **or**, **and** e **not**.

```
In [4]: val1 = True
```

```
In [5]: val2 = False
```

```
In [6]: val1 or val2
```

```
Out[6]: True
```

```
In [7]: val1 and val2
```

```
Out[7]: False
```

```
In [8]: not val1
```

```
Out[8]: False
```

Linguagem Python - Tipos

- Numéricos:
 - Literais: 5, 180L, 3.14, 2 + 4j.

```
In [20]: type(5)
```

```
Out[20]: int
```

```
In [21]: type(180L)
```

```
Out[21]: long
```

```
In [22]: type(3.14)
```

```
Out[22]: float
```

```
In [23]: type(2 + 4j)
```

```
Out[23]: complex
```

Linguagem Python - Tipos

- String:
 - Literais: Podem utilizar aspas simples ou aspas duplas.
 - Python não possui tipo caractere.

```
In [24]: strval1 = 'isto é uma string'
```

```
In [25]: strval2 = "isto também é uma string"
```

```
In [26]: strval3 = u'string unicode'
```

Linguagem Python - Tipos

- String:
 - Concatenação:

```
In [36]: strval_a = "abc"
```

```
In [37]: strval1 = "abc"
```

```
In [38]: strval2 = "def"
```

```
In [39]: strval3 = strval1 + strval2
```

```
In [40]: strval3
```

```
Out[40]: 'abcdef'
```

```
In [41]: print strval3
```

```
abcdef
```

Linguagem Python - Operadores

Operador	Descrição
<	Menor que
<=	Menor ou igual
>	Maior
>=	Maior ou igual
!= ou <>	Diferente
==	Igual
is	Testa se dois objetos são o mesmo objeto.
is not	Testa se dois objetos não são o mesmo objeto.

Linguagem Python - Operadores

- Utilizar com cuidado!!

```
In [53]: a = b = 3
```

```
In [54]: a is b
```

```
Out[54]: True
```

```
In [55]: a is not b
```

```
Out[55]: False
```

```
In [56]: c = 7
```

```
In [57]: a is c
```

```
Out[57]: False
```

Linguagem Python - Operadores

Operador	Descrição
+	Soma
$x - y$	Subtração
*	Produto
/	Quociente
$x // y$	Div
$x \% y$	Mod
$x ** y$	Potência
int(x)	Conversão para inteiro
float(x)	Conversão para ponto flutuante
long(x)	Conversão para long
complex(re, img)	Número complexo
z. conjugate()	Conjugado do número complexo

Linguagem Python – Tuplas e Listas

- Tuplas:

```
In [73]: T = (3, 5)
```

```
In [74]: print T  
(3, 5)
```

```
In [75]: T = (3, 2 + 4j, 'teste')
```

```
In [76]: print T  
(3, (2+4j), 'teste')
```

```
In [77]: a, b, _ = T
```

```
In [78]: print a, b  
3 (2+4j)
```

```
In [79]: T = (a,)
```

```
In [80]: T  
Out[80]: (3,)
```

Linguagem Python – Tuplas e Listas

- Listas:

```
In [86]: L = [ 2, 3, 5 ]
```

```
In [87]: M = [ 4, 6, 7 ]
```

```
In [88]: print L + M  
[2, 3, 5, 4, 6, 7]
```

```
In [89]: L = [ (3, 2), (2, 4), (5, 6) ] # lista de arestas
```

```
In [90]: L  
Out[90]: [(3, 2), (2, 4), (5, 6)]
```

```
In [91]: print sort(L + M)  
[4 6 7 (2, 4) (3, 2) (5, 6)]
```

```
In [92]: len(L)  
Out[92]: 3
```

Linguagem Python – Tuplas e Listas

- Listas:

```
In [93]: L = []
```

```
In [94]: L = [] # lista vazia
```

```
In [95]: L.append(3) # insere objeto no final de L
```

```
In [96]: L.append(4)
```

```
In [97]: L.append([6, 7])
```

```
In [98]: L
```

```
Out[98]: [3, 4, [6, 7]]
```

```
In [99]: L.remove(4) # remove a primeira ocorrencia de 4
```

```
In [100]: L
```

```
Out[100]: [3, [6, 7]]
```

Linguagem Python – Tuplas e Listas

- Tuplas são imutáveis.
 - Não se pode inserir ou remover objetos da mesma tupla.
 - Podemos utilizara para representar arestas.
- Listas são mutáveis.
 - Objetos podem ser inseridos e removidos na mesma lista.
 - Podemos utilizar para representar listas de arestas.

Linguagem Python – Tuplas e Listas

- Acessando elementos da tupla:

```
In [101]: T = (3, 4) # tupla T
```

```
In [102]: T[0]
```

```
Out[102]: 3
```

```
In [103]: T[1]
```

```
Out[103]: 4
```

Linguagem Python – Tuplas e Listas

- Acessando elementos da lista:

```
In [124]: L = [3, 4, 5, 7, 9, 1] # lista L
```

```
In [125]: L[3] # elemento 3 contando a partir de 0
```

```
Out[125]: 7
```

```
In [126]: L[-1] # último elemento
```

```
Out[126]: 1
```

```
In [127]: L[-2] # penúltimo elemento
```

```
Out[127]: 9
```

```
In [128]: L[1:] # do elemento 1 ao fim da lista
```

```
Out[128]: [4, 5, 7, 9, 1]
```

```
In [129]: L[1:3] # de 1 a 3 (o elemento 3 não é incluído)
```

```
Out[129]: [4, 5]
```

```
In [130]: L[:3] # até o elemento 3, não incluindo o elemento 3
```

```
Out[130]: [3, 4, 5]
```


Linguagem Python – Repetição

```
# estrutura de repetição utilizando for
# repetir para i = 0 até 9 (10 - 1)
for i in range(10):
    # indentação é muito importante em Python!
    # imprimir i
    print i

# podemos usar o laço for para iterar os elementos de uma lista
L = [ 3, 5, 8]
# laço for
for l in L:
    # instruções...
    print l

# contador
j = 0
# loop while j < 10
while (j < 10):
    # instruções...
    print j
    # incrementar j
    j = j + 1;
```

Linguagem Python – If

```
if (a > b):  
    print 'a é maior que b'  
elif (a == b):  
    print 'a e b são iguais'  
else:  
    print 'b é maior que a'
```

Linguagem Python – Funções

exemplo de uma função bastante simples

```
def soma(a, b):  
    return a + b;
```

exemplo de uma função um pouco mais complexa

```
def fib(n):  
    """  
    Descrição  
    -----  
    Algoritmo iterativo para a sequência de Fibonacci.  
  
    Entradas  
    -----  
    n : um número inteiro positivo.  
    """  
    i = 1  
    j = 0  
  
    for k in range(n):  
        t = i + j  
        i = j  
        j = t  
  
    return j
```

Linguagem Python – Import

- Em Python podemos importar bibliotecas utilizando o seguinte comando:
 - **import** <module> [as <alias>]
- De modo alternativo, podemos também importar uma função específica
 - **from** <module> **import** <function> [as <alias>]
 - **from** <module> **import** *

Linguagem Python – Import

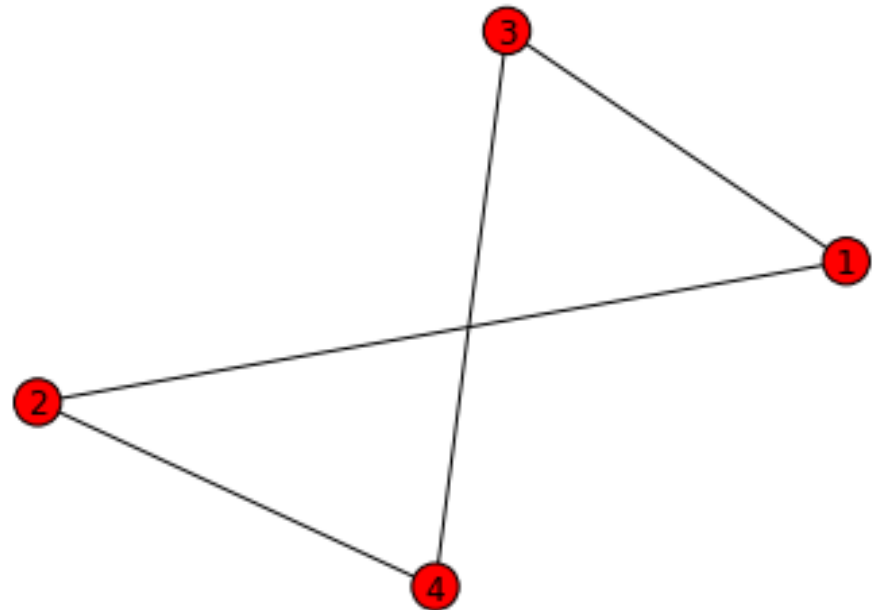
```
In [3]: import networkx as nx # importando biblioteca networkx
```

```
In [4]: G = nx.Graph() # usando a networkx para criar um grafo G
```

```
In [5]: L = [ (1, 2), (1, 3), (3, 4), (2, 4) ] # construir lista de arestas
```

```
In [6]: G.add_edges_from(L) # incluir arestas em G
```

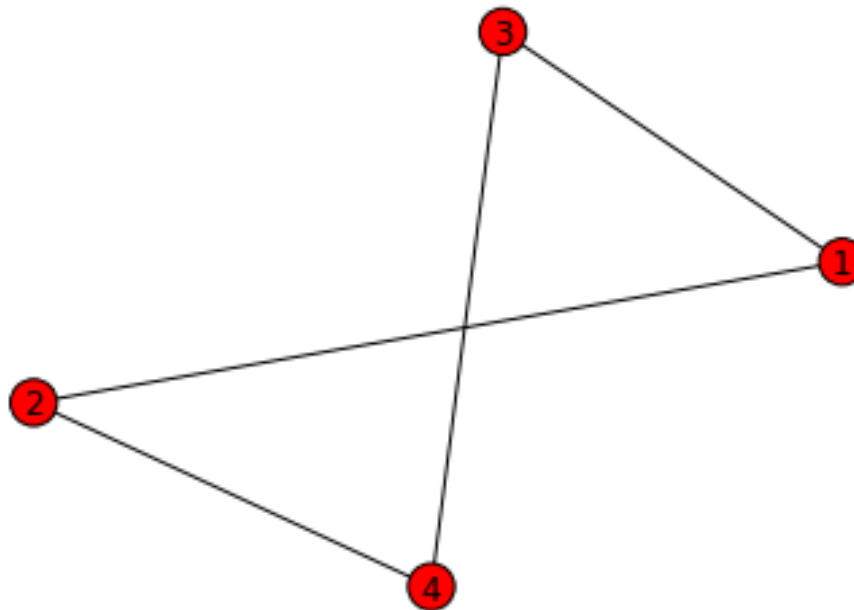
```
In [7]: nx.draw(G)
```



Linguagem Python – Import

```
In [8]: from networkx import draw as draw_graph
```

```
In [9]: draw_graph(G)
```

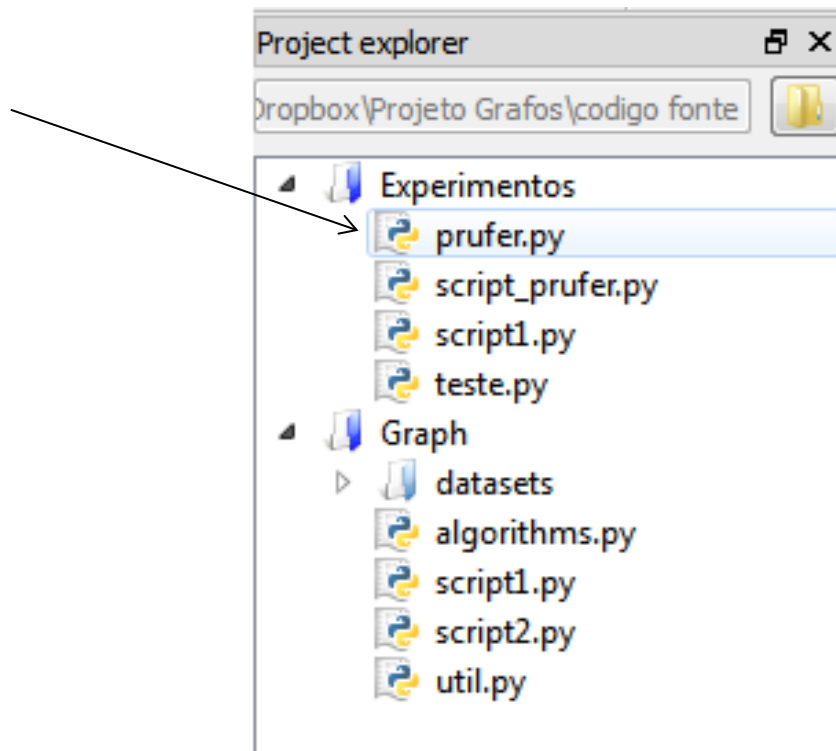


Algoritmo de Prüfer

- Implementaremos cada algoritmo na forma de uma função e os experimentos serão executados criando um script Python.
 - `prufer_encode`:
 - Computa o código de Prüfer dado uma MST.
 - `prufer_decode`:
 - Computa uma árvore dado um uma string contendo o código.

Algoritmo de Prüfer

- Criar um novo módulo chamado “prufer.py” para incluir as nossas funções.



Algoritmo de Prüfer

- Importar os módulos necessários:

```
3 import string  
4 import networkx as nx
```

- Utilizaremos a networkx para manipular os grafos.
 - Podemos chamar funções da networkx utilizando “*nx.alguma_função(...)*”.
- O modulo string será usado para transformar uma lista de rótulos em um código string.

Algoritmo de Prüfer

- Algoritmo de Prüfer fará uso da definição de nó folha de um grafo.
 - **Definição:** Seja $G(V, E)$ um grafo não direcionado, para $v \in V$, v é um nó folha se v possui grau 1, ou seja, $d(v) = 1$.
 - **Notação matemática:**

$$\forall v \in V, v \text{ é folha se } d(v) = 1$$

Algoritmo de Prüfer

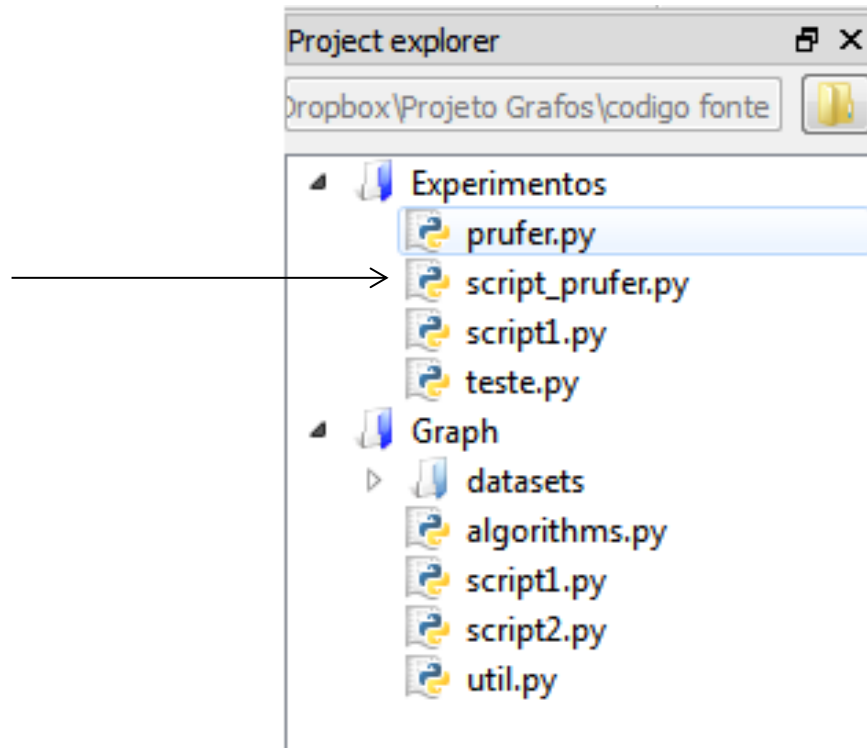
```
6 def prufer_encode(T):
7
8     # iniciando uma lista vazia
9     S = [];
10    # copiar T para não modificar grafo original
11    G = T.copy();
12    # enquanto houver mais de um único vertice
13    while (len(G.nodes()) > 1):
14        # obter as folhas de G
15        leafs = [ v for v, d in G.degree().items() if d == 1 ];
16        # obter a folha de menor rótulo
17        u = min(leafs);
18        # vertice incidente a folha de menor rótulo
19        v = G.neighbors(u)[0]; # possui um único vertice incidente
20        # adicionar o rótulo de v na lista S
21        S.append(v);
22        # remover u de G
23        G.remove_node(u);
24
25    # transforma cada rótulo em S em uma string
26    # e concatena todas as strings
27    return string.join(map(str, S));
```

Algoritmo de Prüfer

```
def prufer_decode(code):  
    # converter a string code em uma lista de inteiros  
    C = map(string.atoi, string.split(code, sep=" "));  
    # número de vertices  
    n = len(C) + 1;  
    # conjunto S dos números de 1 a n {1, 2, ..., n}  
    S = range(1, n + 1);  
    # árvore inicial T  
    T = nx.Graph();  
    # iniciar o conjunto de vertices  $|V| = n$   
    T.add_nodes_from(S);  
  
    # enquanto C não for vazio  
    while (C):  
        # buscar em S o menor rótulo que não pertence a C  
        s_min = min([ s for s in S if s not in C ]);  
        # elemento mais a esquerda de C  
        c_i = C[0];  
        # adicionar a T a aresta definida por (c_i, s_min)  
        T.add_edge(*(c_i, s_min));  
        # remover s_min de S  
        S.remove(s_min);  
        # remover c_i de C  
        C = C[1:];  
  
    return T;
```

Algoritmo de Prüfer

- Criar um novo script chamado “script_prufer.py” para incluir as nossas funções.



Algoritmo de Prüfer

```
3 import networkx as nx
4 import matplotlib.pyplot as plt # utilizada para realizar a plotagem do grafo
5
6 def plot_weighted_graph(G):
7     """
8     Descrição
9     -----
10    Plota um grafo ponderado G.
11    """
12    plt.figure();
13    # obter posição dos nós
14    pos = nx.spring_layout(G);
15    # plotar vertices
16    nx.draw_networkx_nodes(G, pos);
17    # plotar rótulos dos vertices
18    nx.draw_networkx_labels(G, pos);
19    # plotar arestas
20    nx.draw_networkx_edges(G, pos);
21    # obter dicionário com as arestas e seus respectivos pesos
22    weights = { (u, v): data['weight'] for u, v, data in G.edges(data=True) }
23    # plotar rótulos das arestas
24    nx.draw_networkx_edge_labels(
25        G, pos, edge_labels=weights, font_size=12, font_family='sans-serif');
26    # plotar G
27    plt.show();
28    # fechar janela atual
29    plt.close();
```

Algoritmo de Prüfer

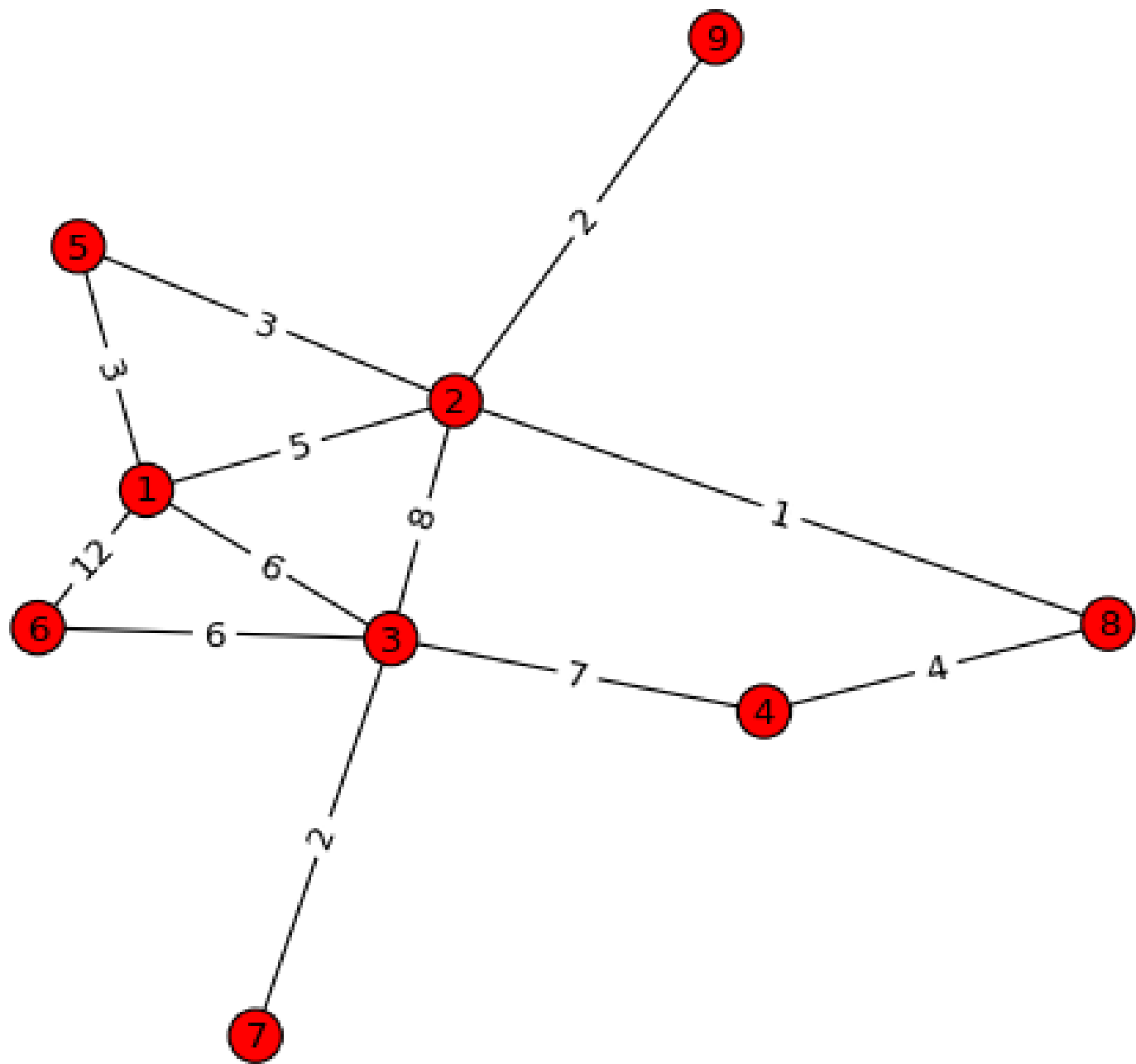
```
32 # criando novo grafo G vazio
33 G = nx.Graph();
34 # adicionando arestas e vertices a G
35 G.add_edge(1, 2, weight=5);
36 G.add_edge(2, 3, weight=8);
37 G.add_edge(1, 3, weight=6);
38 G.add_edge(3, 4, weight=7);
39 G.add_edge(2, 9, weight=2);
40 G.add_edge(5, 2, weight=3);
41 G.add_edge(1, 6, weight=12);
42 G.add_edge(3, 6, weight=6);
43 G.add_edge(3, 7, weight=2);
44 G.add_edge(8, 4, weight=4);
45 G.add_edge(8, 2, weight=1);
46 G.add_edge(1, 5, weight=3);
47 # calcular a MST de G
48 Gmst = nx.minimum_spanning_tree(G);
49 # plotar G
50 plot_weighted_graph(G);
51 # plotar a MST de G
52 plot_weighted_graph(Gmst);
```

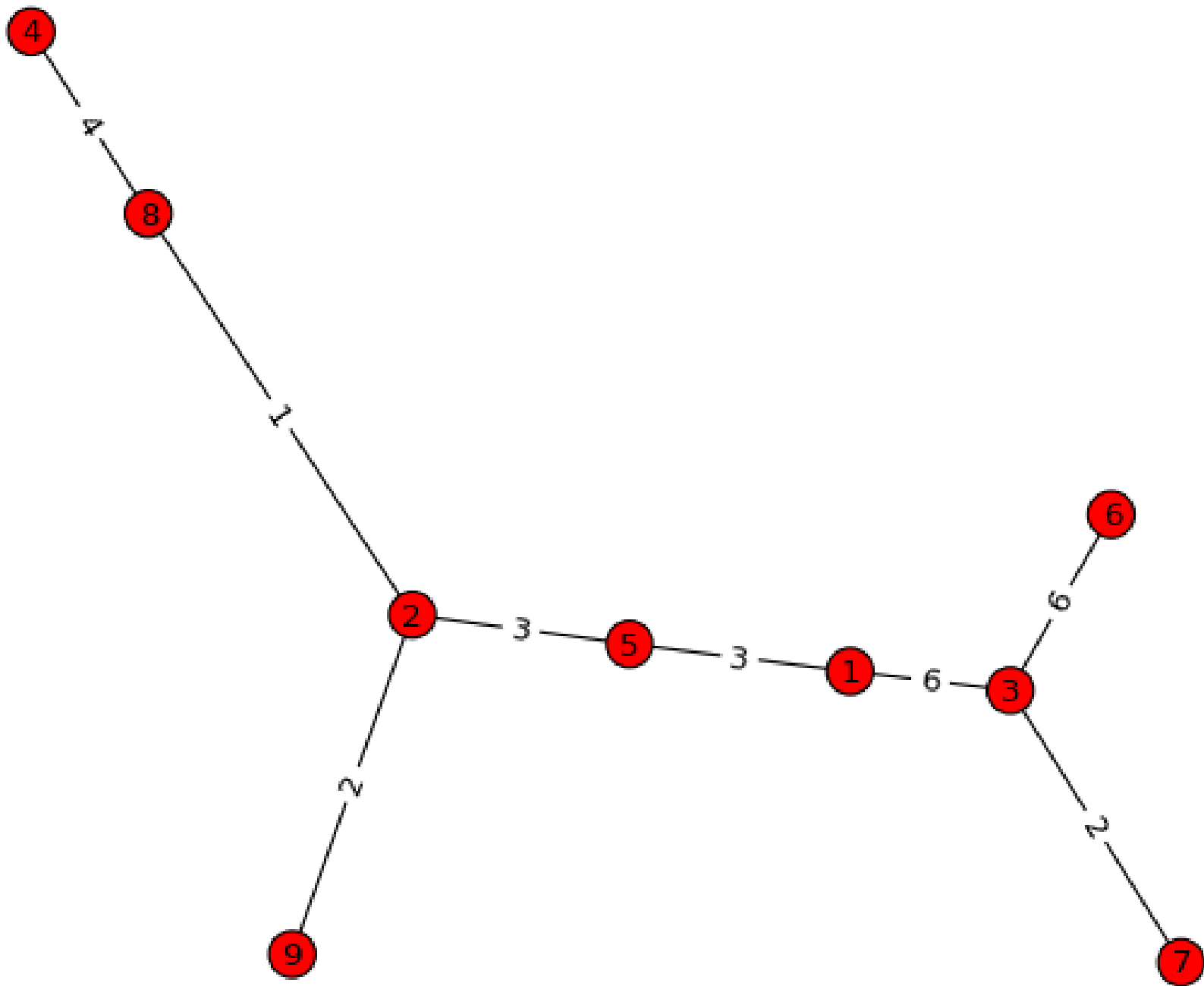
Algoritmo de Prüfer

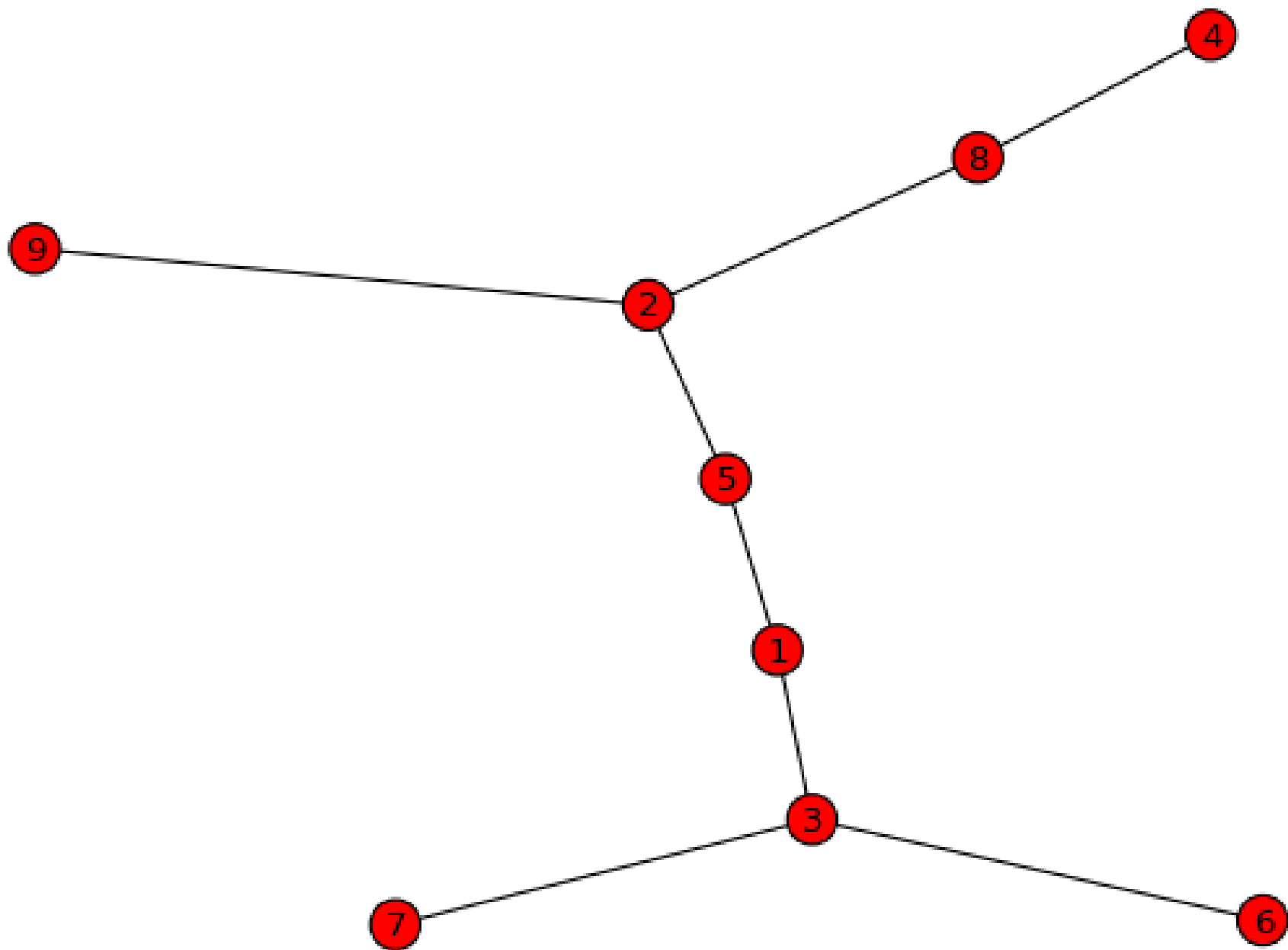
```
54 # importar nossos algoritmos
55 from prufer import prufer_encode
56 from prufer import prufer_decode
57 # gerar o código a partir da MST
58 code = prufer_encode(Gmst);
59 # imprimir o código
60 print code;
61 # obter árvore a partir do código
62 T = prufer_decode(code);
63 # nova figura
64 plt.figure();
65 # plotar árvore
66 nx.draw(T);
67 # exibir
68 plt.show();
69 # fechar janela atual
70 plt.close();
```


Algoritmo de Prüfer

- Para executar o script criado:
 - `%run scprit_prufer.py`
- Código gerado para a MST do exemplo:
 - 8 3 3 1 5 2 2 9
- Seguem os grafos gerados...







Busca em largura

```
3 from collections import deque
4
5 def bfs(G, s):
6
7     # dicionário para armazenar os antecessores
8     P = {} # estrutura de dados que mapeia uma chave a um valor
9     # inicialização do algoritmo
10    for v in G.nodes():
11        G.node[v]['color'] = 'white'
12        G.node[v]['lambda'] = float('inf')
13
14    # iniciar cor da raiz como cinza
15    G.node[s]['color'] = 'gray'
16    # custo para a raiz é 0
17    G.node[s]['lambda'] = 0
18    # iniciar fila Q vazia
19    Q = deque()
20    # inserir nó raiz no início da fila
21    Q.append(s)
22
```

Busca em largura

```
23  # enquanto fila não estiver vazia
24  while (len(Q) > 0):
25      # obter o primeiro elemento da fila
26      u = Q.popleft()
27      # para cada vertice adjacente a u
28      for v in G.neighbors(u):
29          # se v é branco
30          if (G.node[v]['color'] == 'white'):
31              # atualizar custo de v
32              G.node[v]['lambda'] = G.node[u]['lambda'] + 1
33              # adicionar u como antecessor de v
34              P[v] = u
35              # atualizar cor de v
36              G.node[v]['color'] = 'gray'
37              # incluir v em Q
38              Q.append(v)
39
40      # atualizar cor de u
41      G.node[u]['color'] = 'black'
42
43  # retorna a lista de antecessores
44  return P
```

```
3 import networkx as nx
4 import matplotlib.pyplot as plt
5
6 from graphsearch import bfs
7
8 # grafo classico
9 G = nx.petersen_graph()
10 # fazer busca em largura
11 P = bfs(G, 0)
12 # árvore da busca em largura
13 T = nx.Graph()
14 # inserir arestas em T
15 T.add_edges_from([ (u, v) for u, v in P.items() ])
16 # posicionamento do grafo T
17 pos = nx.spring_layout(T)
18 # exibir grafo G
19 plt.figure()          # iniciar figura
20 nx.draw(G, pos)       # plogar grafo G
21 plt.show()            # plotar figura
22 plt.close()           # encerrar figura
23 # iniciar figura
24 plt.figure()
25 # plotar vertices de T
26 nx.draw_networkx_nodes(T, pos, node_size=700)
27 # obter dicionário com as distâncias
28 dist = { v: (v, data['lambda']) for v, data in G.nodes(data=True) }
29 # plotar distâncias
30 nx.draw_networkx_labels(T, pos, labels=dist)
31 # plotar arestas de T
32 nx.draw_networkx_edges(T, pos)
33 # exibir figura
34 plt.show()
35 # encerrar figura
36 plt.close()
```

