Python 图论算法实现工具——NetworkX（1）环境配置及图的创建

1. 前言

前段时间和几位小伙伴一起学习数学建模，其中解决拼接碎纸片问题（2013年全国大学生数学建模B题[1]）时候使用了图的模型，虽然Matlab解决具体的图论问题有很好用的工具箱，但由于组里的小伙伴大多使用Python，所以还是希望能使用Python来解决图论相关的问题（其实主要还是Matlab用的比较菜的缘故）。于是我们发现了Python图论相关的package——NetworkX，在接下来的过程中，我将参考NetworkX2.4官方文档[2]来学习使用NetworkX模块。由于官方文档只有英文版本，本人英语水平有限，若出现错误还望大家指正。

ps:

这个公众号其实20年1月份我就开通了，受到杨老师学习报告的启发，我一开始本着记录一些各领域所学知识，分享给身边需要的人的想法，打算写一些知识分享内容的，后来因为各种原因（主要是没有找到合适的写作排版发布平台，好吧没错就是懒）就暂时搁置了。现在终于等到放假了，不管其他的先搞起来，也希望能通过这一个过程锻炼锻炼自己的写作能力，分享自己对于这个世界的所知所想。

2. NetworkX概述

图1：NetworkX logo

NetworkX是一个免费的Python 软件包（package），主要用于创建、操作和研究复杂网络的结构（structure）、动力学（dynamics）和功能（functions）。

在官方文档[2]中，指出NetworkX的几个亮点：

- 支持图（Graph）、有向图（Digraph）和多图（Multigraph）的数据结构
- 支持许多标准图算法
- 网络结构与分析措施
- 是经典图、随机图和综合网络？（synthetic networks）的生成器
- 节点（Nodes）可以是任意内容（例如：文本、图像、XML记录）
- 边可以保存任意数据（例如：权重、时间序列）
- 免费开源3-clause BSD license

- 提高与现有的C、C++和FORTRAN编写的数值算法和代码的接口

NetworkX诞生于2002年5月。原始版本是由Aric Hagberg，Dan Schult和Pieter Swart在2002年和2003年设计和编写的。首次公开发行是在2005年4月。

3. NetworkX的安装配置

对于熟悉Python开发的小伙伴来说，安装配置当然无脑首选`pip`工具啦~

使用`pip`安装当前版本的`networkx`

```
pip install networkx
```

升级到最新版本，使用`--upgrade`标签：

```
pip install --upgrade networkx
```

注意：使用`pip`工具安装到的是发布版本，如果想安装开发版本可以使用`git`安装，详情可参考官方文档中的说明[3]。

查看本地`networkx`是否成功安装,可在命令提示符中输入：

```
\>pip show networkx
```

如果出现以下内容，则安装成功。

```
Name: networkx
Version: 2.4
Summary: Python package for creating and manipulating graphs and networks
Home-page: http://networkx.github.io/
Author: Aric Hagberg
Author-email: hagberg@lanl.gov
License: BSD
Location: c:\anaconda3\lib\site-packages
Requires: decorator
Required-by: scikit-image
```

（当然你也可以直接`import networkx as nx`试一下）

4. NetworkX创建一个简单的图

接下来，我将参考Networkx2.4官方文档[4]，讲解图的创建

1.创建一个空图

```
import networkx as nx
G = nx.Graph()
```

这时，我们就创建好了一个没有结点（Node）也没有边（Edge）的空图。

感兴趣的话你可以打印一下变量`G`的类型:

```
print(type(G))
```

输出:

```
<class 'networkx.classes.graph.Graph'>
```

复习一下图论中图的定义:

> 定义[5]:
>
> A graph G is an ordered triple $(V(G), E(G), \phi_G)$ consisting of a nonempty set $V(G)$ of vertices ,a set $E(G)$,disjoint from $V(G)$,of edges, and an incidence function ϕ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. If e is an edge and u and v are vertices such that $\phi_G(e) = uv$, then e is said to join u and v; the vertices u and v are called the $ends$ of e.

在NetworkX中，结点可以是任何可以哈希的对象（hashable object）。例如一段文本、一张图片、一个XML对象、另外一个图、自定义的结点对象等。（Python的`None`对象不应用作节点，因为它决定在许多函数中是否分配了可选函数参数，此处函数为Python中的函数概念）

2. 结点（Nodes）

图`G`可以通过一些方式扩充。NetworkX包含许多图生成器函数和以多种格式读写图的工具。

例如我们可以在图中添加一个结点（使用`add_node()`方法）

```
G.add_node(1)
```

添加一个由多个结点组成的列表（使用`add_nodes_from`方法），亦或者添加任意一个由结点构成的可迭代对象。

```
G.add_nodes_from([2, 3])
```

如果你的可迭代对象中每个元素是一个二元组`（node，node_attrribute_dict）`，也可以将这些结点和对应的结点属性一同添加进去。

```
H = nx.path_graph(10)
G.add_nodes_from(H)
```

注意，上述例子添加结点的含义是将图`H`中所有结点添加到图`G`中，如图2所示：

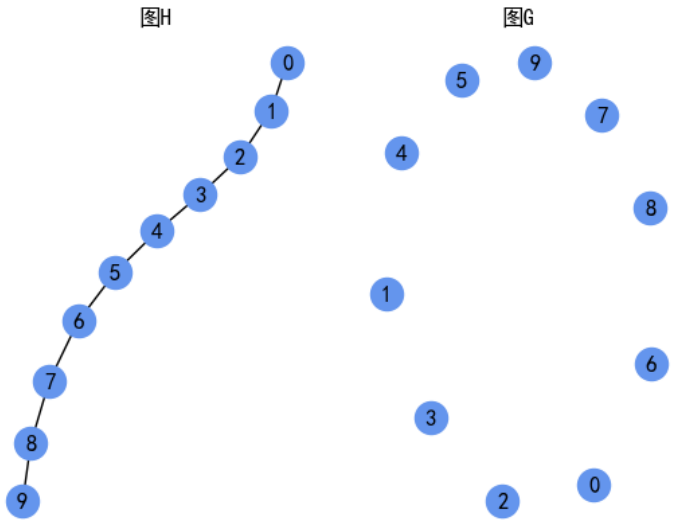

图2：图H与图G的对比

当然，也可以将图`H`做为图`G`的一个结点：

```
G.add_node(H)
```

在具体使用过程中要注意两种方式的区别。

另外的，在官方文档中提到：

> 如果变量的哈希值取决于它的内容，则不应更改结点对象
>
> (You should not change the node object if the hash depends on its contents.)

3. 边（Edges）

图`G`可以通过一次添加一条边来扩充（使用`add_edge()`方法）：

```
G.add_edge(1, 2)
e = (2, 3)
G.add_edge(*e)
# 注意：*e表示一个unpack edge tuple
```

和结点添加一样，图`G`可以由一个由边构成的列表所扩充（使用`add_edges_from()`方法）：

```
G.add_edges_from([(1, 2), (3, 4)])
```

或者通过添加任何的ebunch边[6]。一个ebunch是边元组的任何可迭代对象，这个可迭代对象可以是图`H`。其中，边元组可以是2个节点的元组或3个具有2个节点的三元组，三元组最后一个元素是边属性字典，例如：`(2, 3, {'weight': 3.1415})`，这个例子中边属性字典包含了这条边的权值。那么如果想将ebuch添加到图中：

```
G.add_edges_from(H.edges)
```

如果需要删除图`G`中所有的边，可以使用`clear()`方法

```
G.clear()
```

到现在，我们就可以对一个图`G`添加或删除边和结点了。例如官方文档中的案例：

```
import networkx as nx

G = nx.Graph()
G.add_edges_from([(1, 2), (1, 3)])
G.add_node(1)
G.add_edge(1, 2)
G.add_node("spam")        # 添加一个结点： "spam"
G.add_nodes_from("spam")  # 添加四个结点： 's', 'p', 'a', 'm'
G.add_edge(3, 'm')  # 添加一条结点3与结点'm'的边
```

可视化图`G`结果如图3所示（具体的方法会在NetworkX后续文章中说明）：

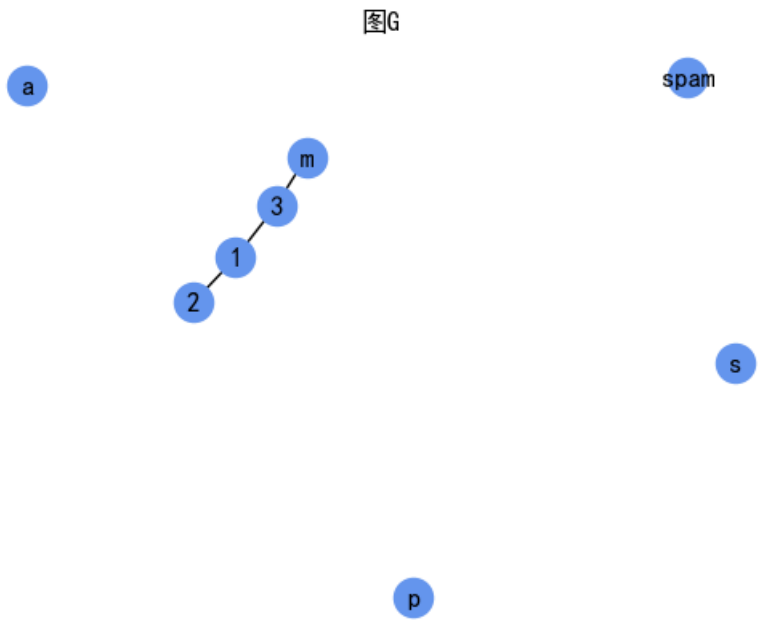

图3：带有结点和边的图G可视化

这个图`G`由8个结点和3条边组成，可利用`number_of_nodes()`、`number_of_edges()`进行考察

```
>>> G.number_of_nodes()
8
>>> G.number_of_edges()
3
```

我们还可以获取图`G`的一些基本信息。比如结点`G.nodes`、边`G.edges`、邻接`G.adj`、结点度`G.degree`，我们还可以将这些图的元素通过如`.item()`、`.data('span')`等方式查看其具体信息。例如：

```
>>> list(G.nodes)
[1, 2, 3, 'spam', 's', 'p', 'a', 'm']
>>> list(G.edges)
[(1, 2), (1, 3), (3, 'm')]
>>> list(G.adj[1])  # or list(G.neighbors(1))
[2, 3]
>>> G.degree[1]  # the number of edges incident to 1
2
```

我们可以使用nbunch[7]来展示边及所有结点的度构成的子集。一个nbunch是任意的：None（即所有节点），节点，或节点中的一个可迭代对象本身不是图中的一个节点。例如：

```
>>> G.edges([2, 'm'])
EdgeDataView([(2, 1), ('m', 3)])
>>> G.degree([2, 3])
DegreeView({2: 1, 3: 2})
```

我们可以采用与添加边或者结点类似的方法来删除图中的结点和边。使用方法:`Graph.remove_node()`、`Graph.remove_nodes_from()`、`Graph.remove_edge()`、`Graph_remove_edges_from()`。例如：

```
>>> G.remove_node(2)
>>> G.remove_nodes_from("spam")
>>> list(G.nodes)
[1, 3, 'spam']
>>> G.remove_edge(1, 3)
```

我们在实例化一种图类来构造图时，可以指定一种格式的数据，例如:

```
>>> G.add_edge(1, 2)
>>> H = nx.DiGraph(G)    # 利用图G来构建一个有向图H
>>> list(H.edges())
[(1, 2), (2, 1)]
>>> edgelist = [(0, 1), (1, 2), (2, 3)]
>>> H = nx.Graph(edgelist)  # 利用edgelist构建一个图H
```

本次的NetworkX工具介绍就到这里啦。如果喜欢这篇内容的话欢迎转发、收藏本文章，您的喜欢是我写作的最大动力！

欢迎关注我的微信公众号：

一位数学专业的在读大学生(菜鸡)

——————————————

生活&音乐&学习&随笔

——————————————

用文字记录平淡生活中每一个值得记录的瞬间。

感谢在茫茫人海中与你相遇。

做点温暖的事情，

愿你也能感受到身边的温暖。

参考资料：

[1] 2013年高教社杯全国大学生数学建模竞赛试题

[2] NetworkX2.4官方文档-index

[3] NetworkX2.4官方文档-install

[4] NetworkX2.4官方文档-creating a graph

[5] Bondy J A , Murty U S R . Graph theory with applications[M]. The Macmillan Press Ltd, 1976.

[6] NetworkX2.4官方文档-reference-glossary-ebunch

[7] NetworkX2.4官方文档-reference-glossary-nbunch