Algoritmos con grafos

November 2, 2023

Introducción a NetworkX

NetworkX es un paquete de Python para crear, manipular y estudiar la estructura de grafos complejos. Ya trae incluidos muchos algoritmos para grafos.

Introducción a NetworkX

NetworkX es un paquete de Python para crear, manipular y estudiar la estructura de grafos complejos. Ya trae incluidos muchos algoritmos para grafos. Como ejemplo básico: acá vemos como crear un grafo:

```
 # casi todo el mundo importa networkx asi
import networkx as nx
G = nx.Graph()
```

Introducción a NetworkX

NetworkX es un paquete de Python para crear, manipular y estudiar la estructura de grafos complejos. Ya trae incluidos muchos algoritmos para grafos. Como ejemplo básico: acá vemos como crear un grafo:

```
 # casi todo el mundo importa networkx asi
import networkx as nx
G = nx.Graph()
```

El grafo G no contiene ningún nodo ni ninguna arista, veamos como agregarlas

Introducción a NetworkX

Podemos agregar nodos de a uno, por ejemplo:

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

Introducción a NetworkX

Podemos agregar nodos de a uno, por ejemplo:

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

o de a varios a la vez

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

Introducción a NetworkX

Podemos agregar nodos de a uno, por ejemplo:

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

o de a varios a la vez

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

Bien! Ahora nuestro grafo tiene nodos, pero aún no tiene aristas, veamos como agregarlas:

Introducción a NetworkX

Podemos agregar aristas de a una, utilizando dos sintaxis distintas:

```
G.add_edge(1, 2)
# o bien
e = (2, 3)
G.add_edge(*e)
```

Introducción a NetworkX

Podemos agregar aristas de a una, utilizando dos sintaxis distintas:

```
G.add_edge(1, 2)
# o bien
e = (2, 3)
G.add_edge(*e)
```

o de a varias a la vez

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

Introducción a NetworkX

Podemos ver cuantos nodos y cuantas aristas tiene nuestro grafo utilizando los métodos asociados:

```
G.number_of_nodes() # 3
G.number_of_edges() # 3
```

Introducción a NetworkX

Podemos ver cuantos nodos y cuantas aristas tiene nuestro grafo utilizando los métodos asociados:

```
G.number_of_nodes() # 3
G.number_of_edges() # 3
```

También podemos obtener la lista completa de nodos y de aristas que contiene un grafo:

```
list(G.nodes)  # [1, 2, 3]
list(G.edges)  # [(1,2), (1, 3), (2, 3)]
```

Introducción a NetworkX

Podemos ver cuantos nodos y cuantas aristas tiene nuestro grafo utilizando los métodos asociados:

```
G.number_of_nodes() # 3
G.number_of_edges() # 3
```

También podemos obtener la lista completa de nodos y de aristas que contiene un grafo:

```
list(G.nodes)  # [1, 2, 3]
list(G.edges)  # [(1,2), (1, 3), (2, 3)]
```

Con el atributo `degree` podemos obtener un diccionario donde las claves son los vértices y los valores asociados son el grado de cada uno de los vértices:

```
dict(G.degree)  # { 1: 2, 2: 2, 3: 3 }
```

Introducción a NetworkX

Del mismo modo que agregamos nodos y aristas, podemos removerlos, utilizando las funciones apropiadas:

```
G.remove_node(2)
G.remove_nodes_from([1,3])
G.remove_edge(1, 3)
```

Introducción a NetworkX

Si queremos que nuestros grafos tengan peso en las aristas, utilizamos el parámetro especial `weight` al momento de agregarla

```
G.add_edge(1, 2, weight=4.7 )
```

Introducción a NetworkX

Una vez que tenemos el grafo listo, podemos empezar a trabajarlo. Por ejemplo, podemos pedirle a NetworkX que analice sus componentes conexas:

```
G = nx.Graph()
G.add_nodes_from([1, 2, 3])
G.add_edges_from([(1, 2), (1, 3)])
G.add_node("spam")        # adds node "spam"
len(list(nx.connected_components(G))) # 2
```

Introducción a NetworkX

Podemos también dibujar el grafo con la ayuda del paquete `matplotlib`

```
import matplotlib.pyplot as plt
G = nx.Graph()
G.add_nodes_from([1, 2, 3])
G.add_edges_from([(1, 2), (1, 3)])
G.add_node("spam")        # adds node "spam"
nx.draw(G, with_labels=True, font_weight='bold')
```

Introducción a NetworkX

Podemos también dibujar el grafo con la ayuda del paquete `matplotlib`

```
import matplotlib.pyplot as plt
G = nx.Graph()
G.add_nodes_from([1, 2, 3])
G.add_edges_from([(1, 2), (1, 3)])
G.add_node("spam")        # adds node "spam"
nx.draw(G, with_labels=True, font_weight='bold')
```

Si necesitamos dibujar grafos con peso, utilizamos la siguiente receta:

```
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold')
edge_labels = dict([
    ((n1, n2), d['weight'])
    for n1, n2, d in G.edges(data=True)
    ])

nx.draw_networkx_edge_labels(G, pos=pos,
   edge_labels=edge_labels)
```

Podemos encontrar la longitud del camino mas corto entre dos vértices utilizando el método `shortest_path_length` y un camino de esa longitud (expresado como una secuencia de vértices) con el método `shortest_path`.

Nota Si en los métodos omitimos el parámetro `weight`, interpretará que todos los pesos son iguales a 1. Para que tome los pesos que asignamos a las aristas, debemos pasar explícitamente `weight="weight"`

```
G = nx.Graph()
G.add_nodes_from("abcdefghijz")
G.add_edge("a", "b", weight=4)
G.add_edge("b", "c", weight=1)
G.add_edge("c", "d", weight=6)
G.add_edge("b", "e", weight=6)
G.add_edge("b", "f", weight=4)
G.add_edge("c", "f", weight=3)
G.add_edge("d", "z", weight=1)
G.add_edge("a", "e", weight=1)
G.add_edge("f", "e", weight=6)
G.add_edge("f", "g", weight=5)
G.add_edge("g", "h", weight=1)
G.add_edge("a", "i", weight=6)
G.add_edge("e", "j", weight=8)
print(nx.shortest_path_length(
    G, source="a", target="z", weight="weight"
))
print(nx.shortest_path(
    G, source="a", target="z", weight="weight"
))
```

Podemos encontrar un árbol de expansión utilizando el algoritmo DFS mediante el método `dfs_tree`.

```
G = nx.Graph()
G.add_nodes_from("abcdef")
G.add_edge("a", "b", weight=4)
G.add_edge("b", "c", weight=1)
G.add_edge("c", "d", weight=6)
G.add_edge("b", "e", weight=6)
G.add_edge("b", "f", weight=4)
G.add_edge("c", "f", weight=3)
G.add_edge("d", "a", weight=1)
G.add_edge("a", "e", weight=1)
G.add_edge("f", "e", weight=6)
G.add_edge("f", "b", weight=5)
G.add_edge("c", "d", weight=1)
G.add_edge("a", "e", weight=6)
G.add_edge("e", "f", weight=8)
T = nx.dfs_tree(G, source='a')
nx.draw(T)
```

Podemos encontrar el árbol de expansión mínima utilizando el método `minimum_spanning_tree`

```
G = nx.Graph()
G.add_nodes_from("abcdef")
G.add_edge("a", "b", weight=4)
G.add_edge("b", "c", weight=1)
G.add_edge("c", "d", weight=6)
G.add_edge("b", "e", weight=6)
G.add_edge("b", "f", weight=4)
G.add_edge("c", "f", weight=3)
G.add_edge("d", "a", weight=1)
G.add_edge("a", "e", weight=1)
G.add_edge("f", "e", weight=6)
G.add_edge("f", "b", weight=5)
G.add_edge("c", "d", weight=1)
G.add_edge("a", "e", weight=6)
G.add_edge("e", "f", weight=8)
T = nx.minimum_spanning_tree(G)
# El peso del arbol T se puede consultar
# con el metodo size
print(T.size(weight="weight"))
```

Referencias

Tutorial oficial de NetworkX

https://networkx.org/documentation/stable/tutorial.html

Lectura recomendada

Johnsonbaugh

Matemáticas discretas. 6ta Edición.

Capítulos 8.5, 9.3 y 9.4

Grimaldi, R.

Matemáticas Discretas y Combinatoria.

Advertencia La terminología asociada a la teoría de grafos no se ha estandarizado aún. Al leer artículos y libros sobre grafos, es necesario verificar las definiciones que se emplean. Ante cualquier duda, consultar con los docentes de la cátedra.