Assignatura: Algorísmica Avançada

Instal·lació de la llibreria NetworkX

A continuació es comenta breument la instal·lació de la llibreria NetworkX pels sistemes operatius Windows i Linux. Cal esmentar que la llibreria té capacitat per dibuixar grafs a pantalla, però que aquesta opció només està disponible a la versió de Linux.

1. **Instal·lació a Windows**. La instal·lació de la llibreria NetworkX a Windows és relativament senzilla. Només cal anar a la pàgina web *http://networkx.lanl.gov/download/networkx*, baixar el fitxer amb extensió `.exe` més nou (`networkx-1.2.linux-i686.exe`) i instal·lar-lo a l'ordinador.
2. **Instal·lació a Linux**. La instal·lació a Linux és més complicada ja que per seguretat el sistema no deixa instal·lar programari fora del nostre directori local de treball. Cal seguir els següents passos per instal·lar NetworkX:
   - Crear un directori a la nostra arrel local (`/home/usuari`), on "usuari" és el nostre usuari. Per exemple, podem crear el directori `/home/usuari/python`,.
   - A continuació cal configurar la variable d'entorn `PYTHONPATH` perquè apunti al directori que acabem de crear. Ho podem fer executant aquesta instrucció a la línia de comandes d'un terminal:

     ```
     export PYTHONPATH=/home/usuari/python
     ```

     És molt recomanable editar el fitxer `~/.bashrc` per incloure-hi al final d'aquest una línia amb

l'anterior instrucció.

- Cal instal·lar a continuació el programa `easy_install`. Per això cal anar al web d'aquest (*http://pypi.python.org/pypi/setuptools*) i baixar-se el fitxer `.egg` que correspon a la versió de Python que hi ha instal·lada a les aules. El fitxer que s'ha de baixar és `setuptools-0.6c11-py2.6.egg`.
- Suposem que el fitxer que hem baixat es troba al directori `~/Baixades`. Executar, des d'un terminal i al directori python que hem creat abans, la següent instrucció per instal·lar el programa `easy_install`

  ```
  sh ~/Baixades/setuptools-0.6c11-py2.6.egg --install-dir=/home/usuari/python
  ```

- La instal·lació de NetworkX és senzilla. Cal situar-se al directory python que hem creat i executar

  ```
  ./easy_install --install-dir=/home/usuari/python networkx
  ```

Abans de continuar amb el següent punt, ens assegurarem que la instal·lació ha estat realitzada de forma correcta. Per això executarem aquestes instruccions des de la línia de comandes de Python:

```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_node("spam")
>>> G.add_edge(1,2)
>>> print(G.nodes())
[1, 2, 'spam']
>>> print(G.edges())
[(1,2)]
```

Si estem fent servir Linux podem comprovar a més les capacitats per dibuixar grafs

```
>>> import networkx as nx
>>> import matplotlib.pyplot as plt
>>> G = nx.petersen_graph()
>>> nx.draw(G)
>>> plt.show()
```

Part 2: Manipulació bàsica de grafs

En aquesta part s'introdueixen les eines bàsiques de manipulació de grafs. En un graf els enllaços són simples i aquests són no dirigits (vegeu *http://networkx.lanl.gov/reference/classes.html* per a altres tipus de grafs disponibles). La informació presentada aquí s'ha estret de la pàgina web de NetworkX, i en particular de les pàgines web *http://networkx.lanl.gov/reference* i *http://networkx.lanl.gov/tutorial*. Es recomana consultar aquestes pàgines web en cas de dubte. Intenteu trobar-hi la informació que busqueu abans de preguntar al professor!

La següent instrucció crea un graf buit.

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

A continuació afegim un vèrtex etiquetat amb el 5 i el 6

```
>>> G.add_node(5)
>>> G.add_node(6)
```

Podem crear un enllaç entre els vèrtexs 5 i 6

```
>>> G.add_edge(5,6)
```

La funció anterior crea els vèrtexs necessaris en cas que no existeixin. Amb la següent instrucció creem dos vèrtexs, 1 i 2, i establim els enllaços indicats.

```
>>> G.add_edge(1,5)
>>> G.add_edge(1,2)
```

Podem consultar informació del graf

```
>>> G.number_of_edges()
3
>>> G.number_of_nodes()
4
```

I podem consultar els enllaços i vèrtexs que hem creat

```
>>> G.edges()
[(1, 2), (1, 5), (5, 6)]
>>> G.nodes()
[1, 2, 5, 6]
```

També podem consultar els veïns d'un vèrtex determinat

```
>>> G.neighbors(5)
[1, 6]
```

Els usuaris de Linux poden, a més, visualitzar el graf en pantalla utilitzant les següents instruccions

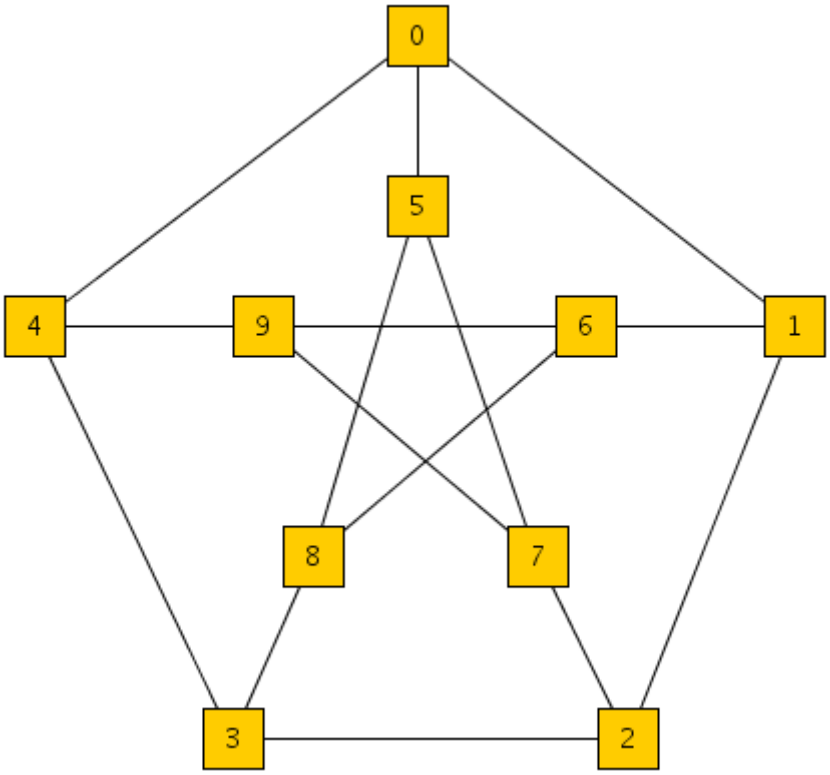
```
>>> import matplotlib.pyplot as plt
>>> nx.draw(G)
>>> plt.show()
```

NetworkX disposa de funcions per llegir i escriure grafs d'un arxiu (a partir de la matriu d'adjacències, per exemple). Podreu trobar més informació al web *http://networkx.lanl.gov/reference/readwrite.html*. Proveu de generar un fitxer d'adjacències i llegiu-lo utilitzant les funcions de la llibreria NetworkX.

La llibreria NetworkX disposa a més de múltiples funcions de manipulació de grafs. Aquestes estan descrites a *http://networkx.lanl.gov/reference/algorithms.html*. Anem a provar l'algorisme DFS sobre un graf predefinit anomenat graf de Petersen (*http://en.wikipedia.org/wiki/Petersen_graph*). L'algorisme DFS és un dels algorismes bàsics de grafs que s'utilitza per explorar els vèrtexs als quals es pot arribar a partir d'un vèrtex inicial.

```
>>> G = nx.petersen_graph()
```

S'inclou a continuació el graf de Petersen tal com el defineix NetworkX.



Anem a analitzar-lo

```
>>> G.number_of_edges()
15
>>> G.number_of_nodes()
10
>>> G.edges()
[(0, 1), (0, 4), (0, 5), (1, 2), (1, 6), (2, 3), (2, 7), (3, 8), (3, 4),
(4, 9), (5, 8), (5, 7), (6, 8), (6, 9), (7, 9)]
>>> G.nodes()
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

L'algorisme DFS és el que hi ha a continuació (correspon a la funció `explore` definida a les transparències del curs d'Algorísmica Avançada). Observeu que es tracta d'un algorisme recursiu que guarda en un graf H és arestes del graf G per les quals va passant.

```
funcio main
      # Sigui G = (Vg,Eg) el graf original
      # Vèrtex inicial d'exploració: v
      # Sigui H = (Vh,Eh) el graf resultant d'aplicar DFS
      # El graf H està buit
      Eh = {}, Vh = {}
      visitats = {}
      explora(v)

funcio explora(u)
      Vh = Vh U {u}
      visitats = visitats U {u}
      per a cada veí k de u no visitat fer:
            Eh = Eh U {(k,u)}
            explora(k)
```

A continuació aplicarem sobre el graf G l'algorisme DFS (Depth First Search) agafant el vèrtex 0 com a inici.

```
>>> H = nx.dfs_tree(G,0)
>>> H.number_of_edges()
9
>>> H.number_of_nodes()
10
>>> H.edges()
[(0, 1), (1, 2), (2, 3), (3, 8), (5, 7), (7, 9), (8, 5), (9, 4), (9, 6)]
>>> H.nodes()
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Observeu les arestes del graf H. Les arestes ens indiquen l'ordre en què el graf G ha estat analitzat (l'ordre d'anàlisi no es pas necessàriament l'ordre en què aquests s'han imprès per pantalla). Observeu també la llista de vèrtexs d'H: és la mateixa que la del graf G original! Això indica que l'algorisme DFS ha estat capaç d'arribar a tots els vèrtexs del graf. Es diu doncs que aquest graf es connex (per definició, un graf és connex si entre cada parell de nodes del graf hi ha un camí que els uneix).

Si haguéssim iniciat l'algorisme DFS per un altre vèrtex l'estructura de graf resultant H serà diferent, però tindrà el mateix nombre de nodes i arestes que a l'exemple anterior.

```
>>> H = nx.dfs_tree(G,5)
>>> H.number_of_edges()
9
>>> H.number_of_nodes()
10
>>> H.edges()
[(0, 1), (1, 2), (2, 3), (3, 8), (5, 0), (6, 9), (8, 6), (9, 4), (9, 7)]
>>> H.nodes()
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Independentment d'on comenci l'algorisme DFS, aquest serà capaç d'arribar a tots els vèrtexs ja que el graf és connex. En particular, l'algorisme DFS es pot utilitzar per comprovar si un graf és connex. Per això anem a crear un graf no connex afegint dos vèrtexs al graf G de Petersen.

```
>>> G.add_edge(1000,1001)
```

Comprovem ara el nombre de vèrtexs del graf

```
>>> G.number_of_nodes()
12
>>> G.nodes()
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1000, 1001]
```

Apliquem-hi l'algorisme de DFS començant pel vèrtex 0

```
>>> H = nx.dfs_tree(G,0)
>>> H.number_of_edges()
9
>>> H.number_of_nodes()
10
>>> H.edges()
[(0, 1), (1, 2), (2, 3), (3, 8), (5, 7), (7, 9), (8, 5), (9, 4), (9, 6)]
>>> H.nodes()
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Atès que els vèrtexs 1000 i 1001 no estan connectats amb la resta de vèrtexs del graf, el DFS no hi arriba (observeu la llista de vèrtexs que forma el graf H). Podem aplicar el DFS a partir del vèrtex 1000.

```
>>> H = nx.dfs_tree(G,1000)
>>> H.number_of_edges()
1
>>> H.number_of_nodes()
2
>>> H.edges()
[(1000, 1001)]
>>> H.nodes()
[1000, 1001]
```

En aquest cas, observeu que l'algorisme només ha estat capaç d'arribar als nodes 1000 i 1001, ja que el vèrtex 1001 està connecta amb el 1000 però aquests dos vèrtexs no estan connectats amb cap vèrtex més. Prova tu mateix de connectar el vèrtex 1000 amb un altre vèrtex del graf de Petersen original i provar que passa en aplicar l'algorisme de DFS.
Un aspecte interessant de la llibreria és la possibilitat d'adjuntar atributs al graf, vèrtexs i arestes (*http://networkx.lanl.gov/tutorial/tutorial.html#adding-attributes-to-graphs-nodes-and-edges*). Per exemple,

```
>>> G = nx.Graph()
>>> G.add_edge(1,2)
>>> G.node[1]['nom']='Barcelona'
>>> G.node[2]['nom']='Girona'
>>> G.edge[1][2]['distancia(km)'] = 150
>>> G.node[1]['nom']
'Barcelona'
>>> G.edge[1][2]['distancia(km)']
150
>>> G.nodes(data=True)
[(1, {'nom': 'Barcelona'}), (2, {'nom': 'Girona'})]
>>> G.edges(data=True)
[(1, 2, {'distancia(km)': 150})]
```

Un atribut utilitzat habitualment en algorismes de grafs (per exemple, Dijkstra, Prim o Kruskal) és el pes (o cost) de l'aresta. Aquest atribut és el `weight` (vegeu el tutorial

A l'hora de programar, la llibreria NetworkX permet accedir molt fàcilment als vèrtexs i arestes del graf (això és gràcies a l'estructura interna de graf utilitzada per NetworkX). Per accedir als vèrtex d'un graf podeu utilitzar aquesta instrucció

```
>>> for n in G:
...    print n
```

Per accedir al veïns d'un vèrtex podeu utilitzar

```
>>> for n in G:
...     for nbr in G[n]:
...        print "El node",n,"te com a vei a",nbr
```