Drawing graphs in Python with networkx

Leave a Comment / Python, Tutorials / By admin

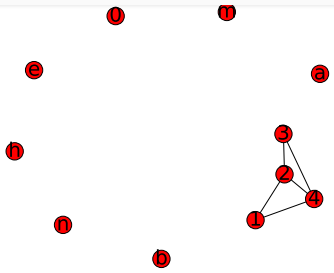
```python
import matplotlib.pyplot as plt

G=nx.Graph()

#########################Adding nodes#########################
G.add_node(1)
G.add_nodes_from([2,3,4])

#Add an nbunch: iterable container of nodes  (e.g. a list, set, graph, file, etc..)
H=nx.path_graph(1)
#G now contains the nodes of H as nodes of G.
G.add_nodes_from(H)
#Graph of Graph
#G.add_node(H)



G.add_node("b")       # adds node "behnam"
G.add_nodes_from("behnam") # adds 6 nodes: 'b','e' ,'h', 'n', 'a','m'


#########################Adding edges#########################
G.add_edge(1, 4)
edge_2_3=(2,3)
G.add_edge(*edge_2_3) # unpack edge tuple with *

#An edge can be associated with any object x using G.add_edge(n1,n2,object=x).
edge_2_4=(2,4,{'weight':3.1415})

G.add_edge(*edge_2_4)

G.add_edge(1, 2, weight=4.7 )
G.add_weighted_edges_from([(3,4,0.125)])


#########################Accessing nodes,edges and neighbors#########################
print 'Nodes'
print G.nodes()
print 'List of edges'
print G.edges()
print 'Neighbors'
print G.neighbors(1)


#Accessing edges
print 'Accessing edges'
print G[3]
print G[4]
print G[4][2]['weight']


for (u,v,d) in G.edges(data='weight'):
    if d>0.5: print('(%d, %d, %.3f)'%(u,v,d))


#########################Adding attributes to nodes#########################
G.add_node(1, time='5pm')
print G.node[1]

G.add_edges_from([(1,2,{'color':'blue'}), (2,3,{'weight':8})])

#########################Multigraphs#########################
#allow you to add the same edge twice, possibly with different edge data.

MG=nx.MultiGraph()
MG.add_weighted_edges_from([(1,2,.5), (1,2,.75), (2,3,.5)])
MG.degree(weight='weight')

GG=nx.Graph()
for n,nbrs in MG.adjacency_iter():
    for nbr,edict in nbrs.items():
        minvalue=min([d['weight'] for d in edict.values()])
        GG.add_edge(n,nbr, weight = minvalue)

print 'shortest path from 1 to 3'
print nx.shortest_path(GG,1,3)



#########################Graph operations#########################
#subgraph(G, nbunch)      - induce subgraph of G on nodes in nbunh
#union(G1,G2)             - graph union
#disjoint_union(G1,G2)    - graph union assuming all nodes are different
#cartesian_product(G1,G2) - return Cartesian product graph
#compose(G1,G2)           - combine graphs identifying nodes common to both
#complement(G)            - graph complement
#create_empty_copy(G)     - return an empty copy of the same graph class
#convert_to_undirected(G) - return an undirected representation of G
#convert_to_directed(G)   - return a directed representation of G

#########################Ploting Graphs #########################


#nx.draw_spectral(G)
#nx.draw_circular(G)
#nx.draw_random(G)

pos=nx.spring_layout(G) # positions for all nodes
nx.draw_networkx_labels(G,pos,font_size=20,font_family='sans-serif')
nx.draw(G,pos)

plt.show()
```

Search ...

Categories

C++
Computer Vision
Docker
Git
Image Processing
Linear Algebra
Machine Learning
Manipulation
Matlab
Python
Robotic
ROS
Signal Processing
Tutorials
Uncategorized

tags



0

Article Rating

← Previous Post

Next Post →

Subscribe

Be the First to Comment!

This site uses Akismet to reduce spam. Learn how your comment data is processed.

0 COMMENTS

computer vision (10) DTW (3)
Dynamic Time Warping (3) eigen (7)
Expectation Maximization (2) Gaussian (2)
Gaussian Mixture Model (3)
Gaussian Mixture Regression (2)
Gazebo (6) git (10) GMM (4)
GMR (2)
histogram of oriented gradient (2)
HOG (2) image processing (15)
laser (3) linear algebra (5)
localization (4)
machine learning (25)
matlab (8) motion planning (4)
Octomap (4) opencv (13)
pcl (14) point cloud (10)
pointcloud (3) pose estimation (5)
projection matrix (3) python (11)
RANSAC (3) ROS (27) SFM (4)
simulation (5) SLAM (4)
Stereo vision (4)
structure from motion (4)
support vector machine (2) SVD (3)
SVM (2) tracking (4)
wheeled robot (4)

About me

Behnam

Software Developer

I'm robotics enthusiastic with several years of experience in software development with C++ and Python. My main areas of interest are machine learning, computer vision, and robotics.

Contacts:

Munich, Germany

behnam[dot]asadi[at]gmail.com