图论模型

NetworkX库

1.导入

`import networkx as nx`

2.图的生成

```
G = nx.Graph()  # 无向图
G = nx.DiGraph()  # 有向图
G = nx.MultiGraph()  # 多重无向图
G = nx.MultiDigraph()  # 多重有向图
```

3.绘图

调用格式：
`draw(G, pos=None, ax=None, **kwds)`

`pos` 表示位置坐标的字典数据，默认为None，可选值：
`circular_layout` 表示顶点在一个圆环上均匀分布；
`random_layout` 表示顶点在一个单位正方形内随机分布；
`shell_layout` 表示顶点在多个同心圆上分布；
`spring_layout` 表示用 Fruchterman-Reingold 算法排列顶点；
`spectral_layout` 表示根据图的拉普拉斯特征向量排列顶点.

4.常用函数

```
import networkx as nx

G = nx.Graph()
G.add_node(1)
G.add_nodes_from(['A', 'B'])
G.add_edge('A', 'B')
G.add_edge(1, 2, weight=0.5)
e = [('A', 'B', 0.3), ('B', 'C', 0.9), ('A', 'C', 0.5), ('C', 'D', 1.2)]
G.add_edge_from(e)
print(G.adj)  # 显示图的邻接表的字典数据
```

```python
print(list(G.adjacency()))  # 显示图的邻接表的列表数据
print(nx.to_numpy_matrix(G))  # 显示图的邻接矩阵
```

最短路径

1.Dijkstra算法

调用格式：

```python
import networkx as nx

path = nx.dijkstra_path(G, source, target, weight='weight')
d = nx.dijkstra_path_length(G, source, target, weight='weight')
```

2.Floyd算法

调用格式：

```python
import networkx as nx

# 返回所有顶点对之间的最短距离矩阵
print(nx.floyd_warshall_numpy(G, nodelist=None, weight='weight'))
```

3.统一调用

```python
import networkx as nx

# method默认为dijkstra，若要使用Floyd算法，设置为bellman-ford
path = shortest_path(G, source=None, target=None, weight=None,
method='dijkstra')
d = shortest_path_length(G, source=None, target=None, weight=None,
method='dijkstra')
```

最小生成树

调用格式：

```python
import networkx as nx

T = nx.minimum_spanning_tree(G, wight='weight', algorithm='kruskal')
```

algorithm用于选择算法，可选值：kruskal，prim，boruvka.
返回值为所求得的最小生成树的可迭代对象.

最大流与最小费用流

1.基本概念

给定一个有向图 $D=(V,\ A)$ ，其中 A 为弧集，在 V 中指定一点，称为发点或者源（记为 v_s）,该点仅发出弧；同时指定一个点称为收点或者汇（记为 v_t），该点仅有进入的弧；其余点称为中间点，对于每一条弧 $(v_i,\ v_j)\in A$ ，对应有一个 $c(v_i,\ v_j)\geq 0$，称为弧的容量。称这样的一个图为一个网络，记为 $D=(V,\ A,\ C)$，其中 $C=\{c_{ij}\}$。

网络上的流，指定义在弧集合 A 上的函数 $f=\{f_{ij}\}=\{f(v_i,\ v_j)\}$，并称 f_{ij} 为弧 $(v_i,\ v_j)$ 上的流量。

满足下列条件的流 f 称为可行流，
(1) 容量限制条件：
对于每一条弧 $(v_i,\ v_j)\in A$，有

$$0\leq f_{ij}\leq c_{ij}$$

(2) 平衡条件：
对于中间点，流入量等于流出量，即对于每个 $i(i\neq s,t)$，有

$$\sum_{j:(v_i,\ v_j)\in A} f_{ij}-\sum_{k:(v_k,\ v_i)\in A} f_{ki}=0$$

对于发点 v_s，有

$$\sum_{j:(v_s,\ v_j)\in A} f_{sj}=v,$$

对于收点 v_t，有

$$\sum_{k:(v_k,\ v_t)\in A} f_{kt}=v,$$

其中 v 为这个可行流的流量，即发点的净输出量。

2.最大流

最大流问题可以转化为下列的线性规划模型：

$$
\begin{aligned}
& max\ v, \\
& s.t. \begin{cases}
\sum\limits_{j:(v_s,\ v_j)\in A} f_{sj} = v, \\
\sum\limits_{j:(v_i,\ v_j)\in A} f_{ij} - \sum\limits_{k:(v_k,\ v_i)\in A} f_{ki} = 0, \quad i \neq s, t, \\
\sum\limits_{k:(v_k,\ v_t)\in A} f_{kt} = v, \\
0 \leq f_{ij} \leq c_{ij}, \quad \forall (v_i, v_j) \in A.
\end{cases}
\end{aligned}
$$

求解调用格式：

```
import networkx as nx

value, flow_dict = nx.maximun_flow(G, s, t, capacity, flow_func=None)
```

其中 s 为发点，t 为汇点， capacity 为边的容量，flow_func 为计算最大流量的函数，返回值 value 为最大流量的值，flow_dict 为流量在每条边上的分配情况。

3.最小费用流

设 b_{ij} 为弧 $(v_i,\ v_j)$ 上的单位费用，则最小费用流可以转化为下列的线性规划模型：

$$
\begin{aligned}
& min \sum_{(v_i,\ v_j)\in A} b_{ij} f_{ij}, \\
& s.t. \begin{cases}
\sum\limits_{j:(v_s,\ v_j)\in A} f_{sj} = v, \\
\sum\limits_{j:(v_i,\ v_j)\in A} f_{ij} - \sum\limits_{k:(v_k,\ v_i)\in A} f_{ki} = 0, \quad i \neq s, t, \\
\sum\limits_{k:(v_k,\ v_t)\in A} f_{kt} = v, \\
0 \leq f_{ij} \leq c_{ij}, \quad \forall (v_i, v_j) \in A.
\end{cases}
\end{aligned}
$$

当 $v = v_{max}$ 时，为最小费用最大流问题；若 $v > v_{max}$ 则本问题无解。

求解调用格式：

```
import networkx as nx

flow_cost, flow_dict = nx.max_flow_min_cost(G, s, t, capacity, cost, 
flow_func)
cost = nx.cost_of_flow(G, flow)
```

其中 s 为发点，t 为汇点， capacity 为边的容量，cost为边上的单位费用，flow_func 为计算最大流量的函数，flow 表示流量，

返回值 flow_cost 为最大流最小费用的值，flow_dict 为流量在每条边上的分配情况，cost表示在 flow流量下的总费用。

关键路径

将路径权重取负，使用Dijkstra或者Floyd算法求出最短路径即可。

若转化为数学规划模型，
设 x_{ij} 为 0-1 变量，当作业 $(i,\ j)$ 位于关键路径上取 1，否则取 0，
则有

$$
\begin{aligned}
& max \sum_{(i,\ j)\in A} t_{ij}x_{ij}, \\
& s.t. \begin{cases}
\sum\limits_{j:(1,\ j)\in A} x_{1j} = 1, \\
\sum\limits_{j:(i,\ j)\in A} x_{ij} - \sum\limits_{k:(k,\ i)\in A} x_{ki} = 0, \quad i \neq 1, n, \\
\sum\limits_{k:(k,\ n)\in A} x_{kn} = 1, \\
x_{ij} = 0\ or\ 1, \quad \forall (i,j) \in A.
\end{cases}
\end{aligned}
$$