```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from networkx.algorithms import bipartite
```

网络类型

无向图

```
G1 = nx.Graph()              #建立一个空的无向图G
G1.add_node(1)               #添加一个节点1
G1.add_edge(2,3)             #添加一条边2-3（隐含着添加了两个节点2、3）
G1.add_edge(3,2)             #对于无向图，边3-2与边2-3被认为是一条边

```

```
G1.nodes()              #输出全部的节点： [1, 2, 3]
G1.edges()              #输出全部的边：[(2, 3)]
G1.number_of_edges()    #输出边的数量：1
```

```
1
```

有向图

```
G2 = nx.DiGraph()            #建立一个空的无向图G
G2.add_node(1)               #添加一个节点1
G2.add_edge(2,3)             #添加一条边2-3（隐含着添加了两个节点2、3）
G2.add_edge(3,2)             #对于无向图，边3-2与边2-3被认为是一条边

```

```
G2.to_undirected()
G1.to_directed()
```

```
<networkx.classes.digraph.DiGraph at 0x1f1f98a0c18>
```

加权图

有向图和无向图都可以给边赋予权重，其中u是起点，v是终点，w是权重

```
G1.add_weighted_edges_from([(0,1,3.0),(1,2,7.5)])
```

```
G1.get_edge_data(1,2)        #输出{'weight': 7.5}，这是一个字典结构
```

```
{'weight': 7.5}
```

网络算法

```
path=nx.all_pairs_shortest_path(G2)      #调用多源最短路径算法，计算图G所有节点间的最短路径
list(path)[0][1]                        #输出节点0、2之间的最短路径序列： [0, 1, 2]
```

```
{1: [1]}
```

度分布

```
G3 = nx.random_graphs.barabasi_albert_graph(1000,3)    #生成一个n=1000，m=3的BA无标度网络
G3.degree(0)                                          #返回某个节点的度
G3.degree()                                           #返回所有节点的度
```

```
DegreeView({0: 8, 1: 30, 2: 42, 3: 88, 4: 71, 5: 32, 6: 57, 7: 108, 8: 34, 9: 24, 10: 24, 11: 48, 12: 37,
13: 65, 14: 56, 15: 25, 16: 20, 17: 20, 18: 42, 19: 7, 20: 10, 21: 39, 22: 23, 23: 18, 24: 11, 25: 14, 26:
20, 27: 27, 28: 17, 29: 28, 30: 10, 31: 36, 32: 13, 33: 12, 34: 12, 35: 15, 36: 11, 37: 10, 38: 9, 39: 18,
40: 9, 41: 15, 42: 4, 43: 10, 44: 11, 45: 12, 46: 17, 47: 16, 48: 10, 49: 4, 50: 19, 51: 13, 52: 5, 53:
32, 54: 12, 55: 13, 56: 22, 57: 12, 58: 5, 59: 10, 60: 15, 61: 10, 62: 21, 63: 17, 64: 7, 65: 3, 66: 13,
67: 8, 68: 13, 69: 7, 70: 6, 71: 14, 72: 17, 73: 24, 74: 9, 75: 11, 76: 8, 77: 9, 78: 5, 79: 3, 80: 5, 81:
3, 82: 17, 83: 8, 84: 9, 85: 20, 86: 11, 87: 9, 88: 9, 89: 6, 90: 3, 91: 19, 92: 5, 93: 6, 94: 24, 95: 9,
96: 7, 97: 9, 98: 3, 99: 11, 100: 7, 101: 9, 102: 5, 103: 10, 104: 7, 105: 13, 106: 6, 107: 11, 108: 12,
109: 10, 110: 8, 111: 10, 112: 9, 113: 7, 114: 8, 115: 11, 116: 3, 117: 10, 118: 6, 119: 13, 120: 6, 121:
11, 122: 6, 123: 14, 124: 19, 125: 11, 126: 5, 127: 5, 128: 10, 129: 4, 130: 7, 131: 21, 132: 13, 133: 8,
134: 5, 135: 8, 136: 4, 137: 4, 138: 9, 139: 8, 140: 10, 141: 10, 142: 8, 143: 9, 144: 7, 145: 4, 146: 11,
147: 6, 148: 5, 149: 15, 150: 5, 151: 8, 152: 7, 153: 4, 154: 8, 155: 8, 156: 11, 157: 17, 158: 7, 159:
10, 160: 10, 161: 6, 162: 6, 163: 3, 164: 4, 165: 11, 166: 9, 167: 3, 168: 7, 169: 8, 170: 4, 171: 11,
172: 6, 173: 6, 174: 8, 175: 5, 176: 8, 177: 11, 178: 7, 179: 7, 180: 4, 181: 8, 182: 4, 183: 13, 184: 6,
185: 13, 186: 10, 187: 6, 188: 6, 189: 7, 190: 5, 191: 4, 192: 7, 193: 6, 194: 6, 195: 3, 196: 6, 197: 6,
198: 8, 199: 8, 200: 4, 201: 7, 202: 5, 203: 12, 204: 8, 205: 4, 206: 5, 207: 5, 208: 5, 209: 12, 210: 5,
211: 3, 212: 6, 213: 10, 214: 5, 215: 4, 216: 8, 217: 7, 218: 7, 219: 7, 220: 3, 221: 6, 222: 8, 223: 8,
224: 6, 225: 6, 226: 6, 227: 8, 228: 4, 229: 4, 230: 4, 231: 8, 232: 4, 233: 5, 234: 3, 235: 3, 236: 5,
237: 4, 238: 4, 239: 5, 240: 6, 241: 8, 242: 5, 243: 6, 244: 7, 245: 3, 246: 5, 247: 4, 248: 15, 249: 9,
250: 4, 251: 6, 252: 8, 253: 7, 254: 6, 255: 7, 256: 5, 257: 8, 258: 5, 259: 7, 260: 4, 261: 6, 262: 4,
263: 8, 264: 3, 265: 3, 266: 4, 267: 4, 268: 6, 269: 5, 270: 6, 271: 7, 272: 7, 273: 4, 274: 5, 275: 4,
276: 3, 277: 9, 278: 5, 279: 4, 280: 5, 281: 4, 282: 8, 283: 6, 284: 10, 285: 7, 286: 6, 287: 3, 288: 5,
289: 6, 290: 4, 291: 4, 292: 6, 293: 4, 294: 8, 295: 7, 296: 5, 297: 3, 298: 7, 299: 6, 300: 5, 301: 4,
302: 5, 303: 8, 304: 4, 305: 4, 306: 4, 307: 3, 308: 8, 309: 4, 310: 5, 311: 5, 312: 6, 313: 4, 314: 4,
315: 5, 316: 7, 317: 3, 318: 4, 319: 9, 320: 8, 321: 4, 322: 6, 323: 5, 324: 5, 325: 6, 326: 4, 327: 4,
328: 7, 329: 3, 330: 3, 331: 4, 332: 4, 333: 5, 334: 7, 335: 8, 336: 7, 337: 5, 338: 5, 339: 5, 340: 12,
341: 6, 342: 5, 343: 8, 344: 8, 345: 4, 346: 6, 347: 4, 348: 5, 349: 5, 350: 3, 351: 7, 352: 4, 353: 4,
354: 8, 355: 7, 356: 4, 357: 5, 358: 3, 359: 5, 360: 3, 361: 4, 362: 4, 363: 4, 364: 7, 365: 4, 366: 7,
367: 5, 368: 4, 369: 6, 370: 4, 371: 7, 372: 7, 373: 5, 374: 3, 375: 3, 376: 5, 377: 7, 378: 4, 379: 4,
380: 5, 381: 6, 382: 6, 383: 5, 384: 4, 385: 7, 386: 4, 387: 4, 388: 6, 389: 3, 390: 4, 391: 5, 392: 4,
393: 5, 394: 3, 395: 3, 396: 4, 397: 4, 398: 4, 399: 3, 400: 5, 401: 4, 402: 3, 403: 4, 404: 4, 405: 5,
406: 3, 407: 4, 408: 5, 409: 4, 410: 6, 411: 3, 412: 5, 413: 5, 414: 6, 415: 9, 416: 3, 417: 5, 418: 5,
419: 5, 420: 4, 421: 4, 422: 4, 423: 5, 424: 3, 425: 3, 426: 3, 427: 3, 428: 4, 429: 8, 430: 8, 431: 8,
432: 3, 433: 7, 434: 4, 435: 5, 436: 3, 437: 4, 438: 4, 439: 8, 440: 5, 441: 5, 442: 3, 443: 3, 444: 9,
445: 4, 446: 6, 447: 10, 448: 3, 449: 6, 450: 3, 451: 3, 452: 6, 453: 3, 454: 4, 455: 4, 456: 4, 457: 4,
458: 5, 459: 3, 460: 3, 461: 6, 462: 4, 463: 4, 464: 3, 465: 6, 466: 3, 467: 5, 468: 3, 469: 5, 470: 5,
471: 4, 472: 4, 473: 3, 474: 5, 475: 3, 476: 3, 477: 5, 478: 6, 479: 5, 480: 3, 481: 4, 482: 3, 483: 5,
484: 4, 485: 3, 486: 7, 487: 7, 488: 5, 489: 5, 490: 5, 491: 3, 492: 4, 493: 6, 494: 6, 495: 4, 496: 6,
497: 4, 498: 4, 499: 7, 500: 5, 501: 3, 502: 5, 503: 6, 504: 5, 505: 4, 506: 5, 507: 3, 508: 7, 509: 4,
510: 4, 511: 4, 512: 3, 513: 4, 514: 4, 515: 5, 516: 9, 517: 3, 518: 5, 519: 4, 520: 3, 521: 3, 522: 4,
523: 4, 524: 4, 525: 3, 526: 6, 527: 4, 528: 3, 529: 3, 530: 3, 531: 5, 532: 6, 533: 4, 534: 3, 535: 4,
536: 5, 537: 4, 538: 5, 539: 4, 540: 4, 541: 3, 542: 5, 543: 3, 544: 3, 545: 3, 546: 3, 547: 3, 548: 3,
549: 6, 550: 3, 551: 5, 552: 4, 553: 4, 554: 3, 555: 3, 556: 5, 557: 4, 558: 3, 559: 6, 560: 4, 561: 3,
562: 4, 563: 5, 564: 3, 565: 3, 566: 4, 567: 5, 568: 4, 569: 4, 570: 4, 571: 3, 572: 5, 573: 4, 574: 3,
575: 4, 576: 5, 577: 3, 578: 4, 579: 5, 580: 3, 581: 5, 582: 4, 583: 4, 584: 4, 585: 4, 586: 6, 587: 4,
588: 3, 589: 3, 590: 4, 591: 4, 592: 3, 593: 3, 594: 4, 595: 5, 596: 3, 597: 3, 598: 4, 599: 8, 600: 6,
601: 4, 602: 3, 603: 5, 604: 3, 605: 3, 606: 7, 607: 3, 608: 3, 609: 3, 610: 3, 611: 4, 612: 5, 613: 4,
614: 3, 615: 6, 616: 4, 617: 4, 618: 4, 619: 4, 620: 5, 621: 4, 622: 7, 623: 3, 624: 3, 625: 4, 626: 3,
627: 4, 628: 4, 629: 3, 630: 6, 631: 3, 632: 4, 633: 4, 634: 4, 635: 3, 636: 3, 637: 3, 638: 4, 639: 7,
640: 3, 641: 3, 642: 3, 643: 3, 644: 3, 645: 4, 646: 6, 647: 5, 648: 3, 649: 4, 650: 3, 651: 4, 652: 3,
653: 4, 654: 3, 655: 3, 656: 3, 657: 5, 658: 3, 659: 4, 660: 4, 661: 3, 662: 3, 663: 3, 664: 3, 665: 4,
```

```
666: 5, 667: 3, 668: 3, 669: 4, 670: 3, 671: 3, 672: 4, 673: 3, 674: 3, 675: 3, 676: 5, 677: 4, 678: 3,
679: 4, 680: 3, 681: 4, 682: 6, 683: 4, 684: 4, 685: 4, 686: 3, 687: 4, 688: 3, 689: 4, 690: 4, 691: 4,
692: 4, 693: 3, 694: 5, 695: 4, 696: 3, 697: 4, 698: 5, 699: 6, 700: 4, 701: 3, 702: 3, 703: 4, 704: 3,
705: 7, 706: 3, 707: 3, 708: 5, 709: 3, 710: 4, 711: 3, 712: 4, 713: 4, 714: 4, 715: 4, 716: 4, 717: 4,
718: 3, 719: 3, 720: 5, 721: 3, 722: 5, 723: 5, 724: 3, 725: 3, 726: 3, 727: 4, 728: 3, 729: 4, 730: 3,
731: 3, 732: 4, 733: 3, 734: 3, 735: 3, 736: 4, 737: 4, 738: 3, 739: 4, 740: 3, 741: 3, 742: 3, 743: 4,
744: 4, 745: 3, 746: 3, 747: 4, 748: 6, 749: 3, 750: 3, 751: 3, 752: 3, 753: 5, 754: 3, 755: 4, 756: 4,
757: 3, 758: 3, 759: 4, 760: 3, 761: 4, 762: 3, 763: 3, 764: 5, 765: 5, 766: 3, 767: 3, 768: 3, 769: 4,
770: 3, 771: 3, 772: 3, 773: 4, 774: 3, 775: 3, 776: 3, 777: 6, 778: 3, 779: 4, 780: 3, 781: 3, 782: 3,
783: 3, 784: 3, 785: 3, 786: 5, 787: 3, 788: 4, 789: 6, 790: 3, 791: 3, 792: 4, 793: 3, 794: 5, 795: 3,
796: 3, 797: 5, 798: 3, 799: 3, 800: 4, 801: 3, 802: 3, 803: 3, 804: 3, 805: 3, 806: 3, 807: 3, 808: 3,
809: 3, 810: 3, 811: 3, 812: 3, 813: 3, 814: 3, 815: 4, 816: 5, 817: 4, 818: 3, 819: 3, 820: 3, 821: 3,
822: 4, 823: 3, 824: 4, 825: 3, 826: 3, 827: 4, 828: 3, 829: 4, 830: 4, 831: 3, 832: 3, 833: 3, 834: 3,
835: 3, 836: 3, 837: 3, 838: 4, 839: 3, 840: 3, 841: 4, 842: 3, 843: 3, 844: 3, 845: 3, 846: 3, 847: 3,
848: 3, 849: 3, 850: 3, 851: 3, 852: 3, 853: 3, 854: 3, 855: 3, 856: 3, 857: 3, 858: 3, 859: 3, 860: 3,
861: 3, 862: 3, 863: 4, 864: 3, 865: 3, 866: 3, 867: 3, 868: 3, 869: 3, 870: 3, 871: 3, 872: 3, 873: 3,
874: 3, 875: 3, 876: 3, 877: 3, 878: 3, 879: 3, 880: 3, 881: 3, 882: 4, 883: 3, 884: 4, 885: 3, 886: 3,
887: 4, 888: 3, 889: 3, 890: 3, 891: 3, 892: 3, 893: 3, 894: 3, 895: 3, 896: 3, 897: 3, 898: 3, 899: 5,
900: 3, 901: 4, 902: 3, 903: 3, 904: 3, 905: 3, 906: 3, 907: 3, 908: 5, 909: 3, 910: 3, 911: 3, 912: 3,
913: 3, 914: 3, 915: 3, 916: 3, 917: 3, 918: 3, 919: 3, 920: 3, 921: 3, 922: 4, 923: 3, 924: 3, 925: 3,
926: 3, 927: 3, 928: 3, 929: 3, 930: 3, 931: 3, 932: 4, 933: 3, 934: 4, 935: 4, 936: 3, 937: 3, 938: 3,
939: 3, 940: 3, 941: 3, 942: 4, 943: 3, 944: 4, 945: 3, 946: 3, 947: 4, 948: 3, 949: 3, 950: 3, 951: 3,
952: 3, 953: 3, 954: 3, 955: 3, 956: 3, 957: 3, 958: 3, 959: 4, 960: 3, 961: 3, 962: 3, 963: 3, 964: 3,
965: 3, 966: 3, 967: 4, 968: 3, 969: 3, 970: 3, 971: 3, 972: 3, 973: 3, 974: 3, 975: 3, 976: 3, 977: 3,
978: 3, 979: 3, 980: 3, 981: 3, 982: 3, 983: 3, 984: 3, 985: 3, 986: 3, 987: 3, 988: 3, 989: 3, 990: 3,
991: 3, 992: 3, 993: 3, 994: 3, 995: 3, 996: 3, 997: 3, 998: 3, 999: 3})
```

```
nx.degree_histogram(G3)     #返回图中所有节点的度分布序列（从1至最大度的出现频次）
```

```
[0,
 0,
 0,
 372,
 220,
 119,
 71,
 50,
 43,
 20,
 20,
 15,
 9,
 10,
 3,
 5,
 1,
 6,
 2,
```

```
3,
4,
2,
1,
1,
4,
1,
0,
1,
1,
0,
1,
0,
2,
0,
1,
0,
1,
1,
0,
1,
0,
0,
2,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
```

```
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1]
```

群聚系数

```
nx.average_clustering(G3)
```

```
0.03546834680082134
```

直径和平均距离

```
nx.diameter(G3)##最长最短路径的长度
```

```
6
```

```
nx.average_shortest_path_length(G3)# 则返回图G所有节点间平均最短路径长度。
```

```
3.4706426426426424
```

匹配性

```
nx.degree_assortativity_coefficient(G3)
```

```
-0.0898140577205234
```

中心性

Degree centrality measures.（点度中心性？）
degree_centrality(G) Compute the degree centrality for nodes.
in_degree_centrality(G) Compute the in-degree centrality for nodes.
out_degree_centrality(G) Compute the out-degree centrality for nodes.

Closeness centrality measures.（紧密中心性？）
closeness_centrality(G[, v, weighted_edges]) Compute closeness centrality for nodes.

Betweenness centrality measures.（介数中心性？）
betweenness_centrality(G[, normalized, ...]) Compute betweenness centrality for nodes.
edge_betweenness_centrality(G[, normalized, ...]) Compute betweenness centrality for edges.

Current-flow closeness centrality measures.（流紧密中心性？）
current_flow_closeness_centrality(G[, ...]) Compute current-flow closeness centrality for nodes.
Current-Flow Betweenness

Current-flow betweenness centrality measures.（流介数中心性？）
current_flow_betweenness_centrality(G[, ...]) Compute current-flow betweenness centrality for nodes.
edge_current_flow_betweenness_centrality(G) Compute current-flow betweenness centrality for edges.

Eigenvector centrality.（特征向量中心性？）
eigenvector_centrality(G[, max_iter, tol, ...]) Compute the eigenvector centrality for the graph G.
eigenvector_centrality_numpy(G) Compute the eigenvector centrality for the graph G.

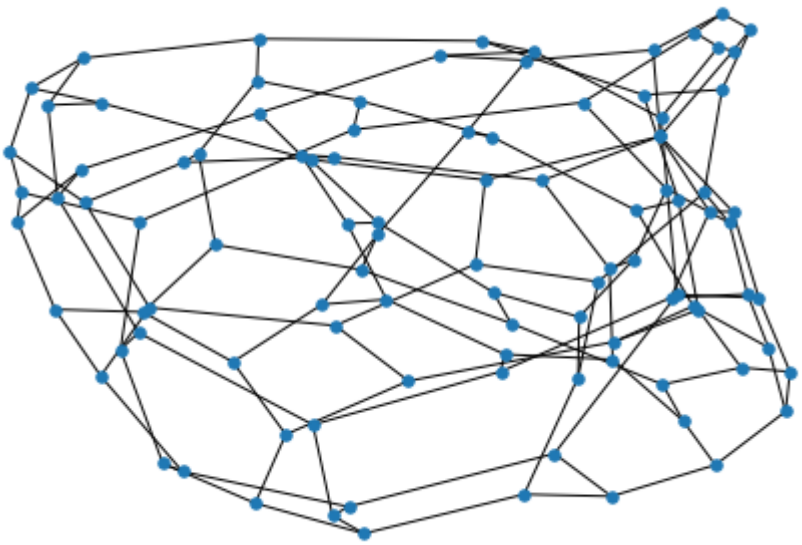
Load centrality.
load_centrality(G[, v, cutoff, normalized, ...]) Compute load centrality for nodes.
edge_load(G[, nodes, cutoff]) Compute edge load.

网络模型

规则网络

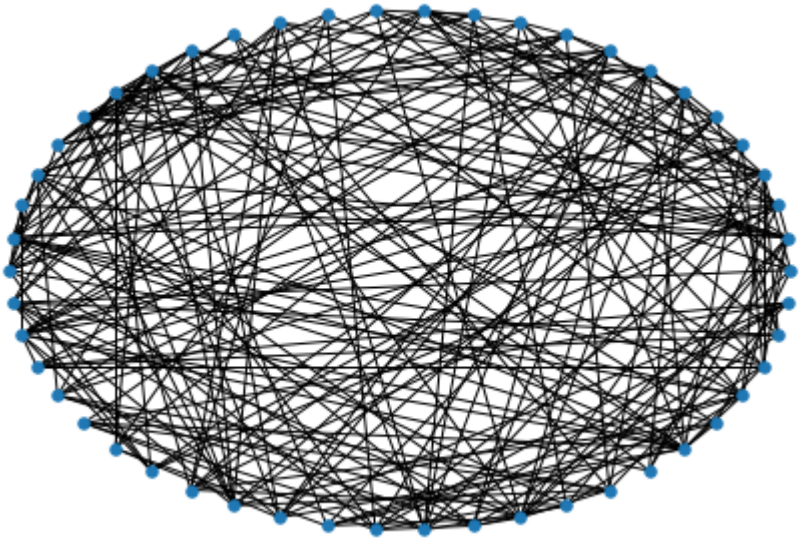
```
RG = nx.random_graphs.random_regular_graph(3,100)  #生成包含20个节点、每个节点有3个邻居的规则图RG
pos = nx.spectral_layout(RG)           #定义一个布局，此处采用了spectral布局方式，后变还会介绍其它布局方式，注意图形上的区别
nx.draw(RG,pos,with_labels=False,node_size = 30)  #绘制规则图的图形，with_labels决定节点是非带标签（编号），node_size是节点的直径

```



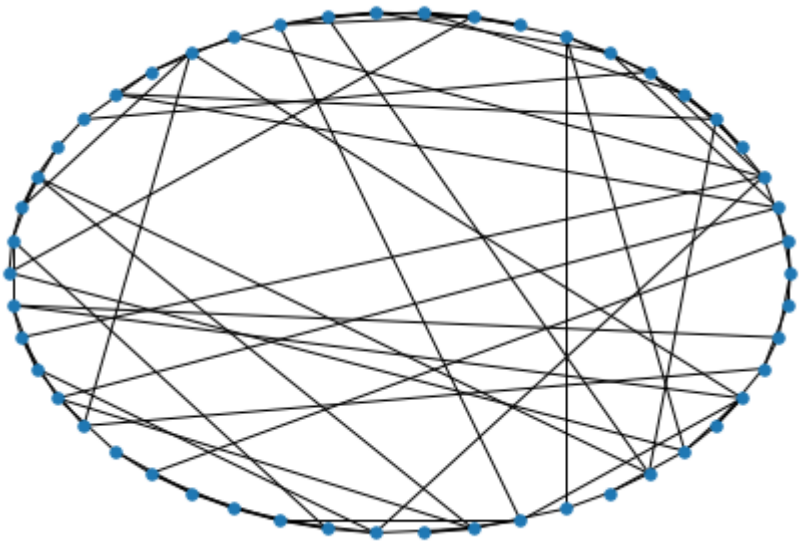
ER随机图

```
ER = nx.random_graphs.erdos_renyi_graph(50,0.2)  #生成包含20个节点、以概率0.2连接的随机图
pos = nx.shell_layout(ER)          #定义一个布局，此处采用了shell布局方式
nx.draw(ER,pos,with_labels=False,node_size = 30)
plt.show()
```



WS小世界网络

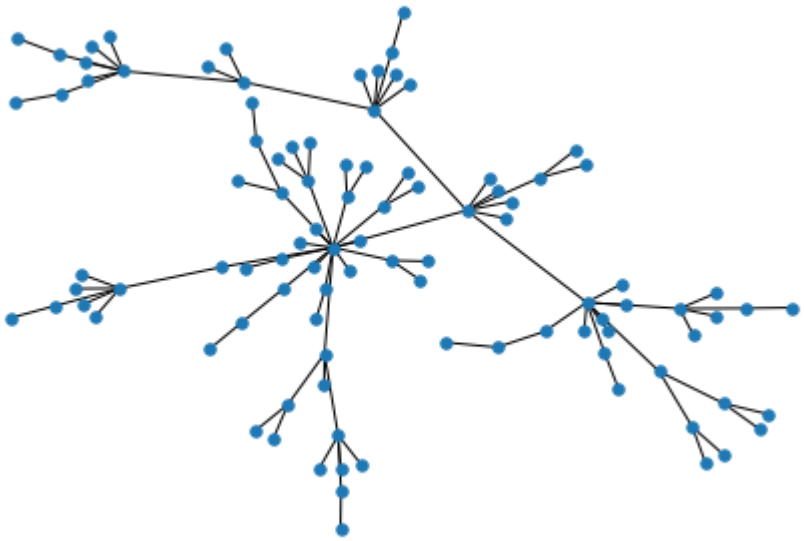
```
WS = nx.random_graphs.watts_strogatz_graph(50,4,0.3)  #生成包含20个节点、每个节点4个近邻、随机化重连概率为0.3的小世界网络
pos = nx.circular_layout(WS)           #定义一个布局，此处采用了circular布局方式
nx.draw(WS,pos,with_labels=False,node_size = 30)  #绘制图形
plt.show()
```



BA无标度网络

```
BA= nx.random_graphs.barabasi_albert_graph(100,1)  #生成n=20、m=1的BA无标度网络
pos = nx.spring_layout(BA)           #定义一个布局，此处采用了spring布局方式
nx.draw(BA,pos,with_labels=False,node_size = 30)  #绘制图形

```



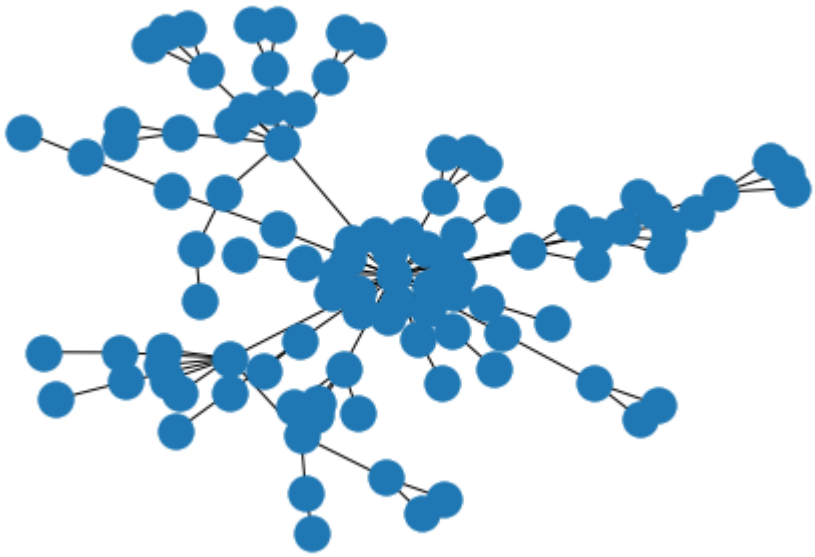
BA代码

```
#定义一个方法，它有两个参数：n - 网络节点数量；m - 每步演化加入的边数量
def barabasi_albert_graph(n, m):
    # 生成一个包含m个节点的空图（即BA模型中t=0时的m0个节点）
    G=empty_graph(m)
    # 定义新加入边要连接的m个目标节点
    targets=range(m)
    # 将现有节点按正比于其度的次数加入到一个数组中，初始化时的m个节点度均为0，所以数组为空
    repeated_nodes=[]
    # 添加其余的 n-m 个节点，第一个节点编号为m（Python的数组编号从0开始）
    source=m
    # 循环添加节点
    while source<n:
        # 从源节点连接m条边到选定的m个节点targets上（注意targets是上一步生成的）
        G.add_edges_from(zip([source]*m,targets))
        # 对于每个被选择的节点，将它们加入到repeated_nodes数组中（它们的度增加了1）
        repeated_nodes.extend(targets)
        # 将源点m次加入到repeated_nodes数组中（它的度增加了m）
        repeated_nodes.extend([source]*m)
        # 从现有节点中选取m个节点 ，按正比于度的概率（即度优先连接）
        targets=set()
        while len(targets)<m:
            #按正比于度的概率随机选择一个节点，见注释1
            x=random.choice(repeated_nodes)
            #将其添加到目标节点数组targets中
            targets.add(x)
        #挑选下一个源点，转到循环开始，直到达到给定的节点数n
        source += 1
    #返回所得的图G
    return G
```

网络可视化

基本流程

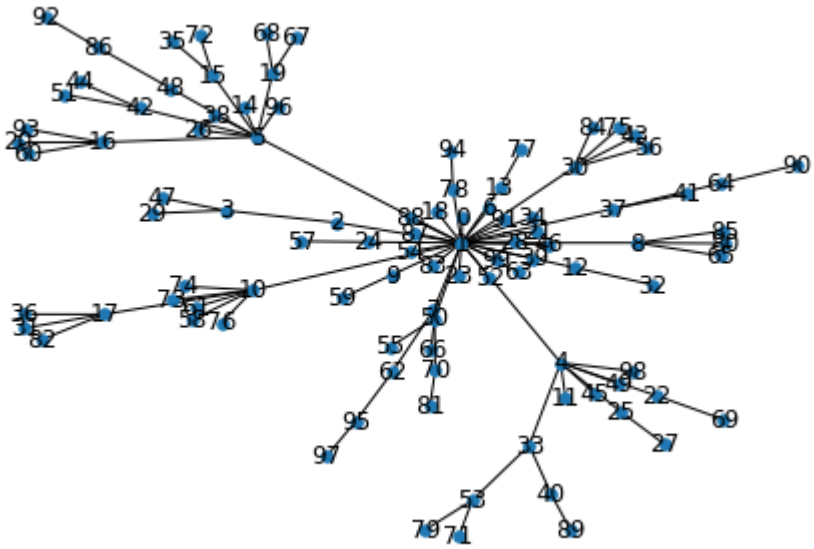
```
G4 =nx.random_graphs.barabasi_albert_graph(100,1)    #生成一个BA无标度网络G
nx.draw(G4)                                 #绘制网络G
```



样式

```
  - `node_size`:  指定节点的尺寸大小(默认是300，单位未知，就是上图中那么大的点)
  - `node_color`:  指定节点的颜色 (默认是红色，可以用字符串简单标识颜色，例如'r'为红色，'b'为绿色等，具体可查看手册)
  - `node_shape`:  节点的形状（默认是圆形，用字符串'o'标识，具体可查看手册）
  - `alpha`: 透明度 (默认是1.0，不透明，0为完全透明)
  - `width`: 边的宽度 (默认为1.0)
  - `edge_color`: 边的颜色(默认为黑色)
  - `style`: 边的样式(默认为实现，可选:  solid|dashed|dotted,dashdot)
  - `with_labels`: 节点是否带标签（默认为True）
  - `font_size`: 节点标签字体大小 (默认为12)
  - `font_color`: 节点标签字体颜色（默认为黑色）
```

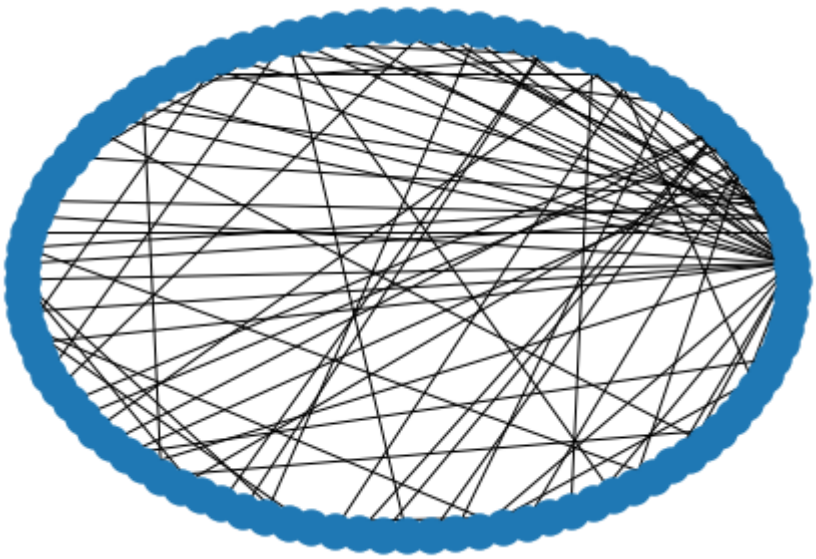
```
nx.draw(G4,node_size = 30,with_labels =True)
```



布局

circular_layout：节点在一个圆环上均匀分布
random_layout：节点随机分布
shell_layout：节点在同心圆上分布
spring_layout： 用Fruchterman-Reingold算法排列节点
spectral_layout：根据图的拉普拉斯特征向量排列节点？我也不是太明白

```
1 nx.draw(G4,pos = nx.circular_layout(G4))
```



文本

二分图

二分图又称二部图，是图论中的一种特殊模型，它的顶点可分割为两个互不相交的子集，并且图中的每条边所关联的两个顶点分别属于这两个不同的顶点集

构建二分图

例如科学家合作网络（作者和论文）、商品网络（商品和购买者）、城市公交网络（线路和站点）

```
B = nx.Graph()
#添加一个项目101，它有3个参与者：201,202,203
B.add_edge(101,201)
B.add_edge(101,202)
B.add_edge(101,203)
#添加一个项目102，它有2个参与者：203,202,2034
B.add_edge(102,203)
B.add_edge(102,204)
```

NetworkX还提供了多种二分图演化模型的建立方法，在这里把它们列出来供大家参考：
-- networkx.generators.classic.complete_bipartite_graph(n1, n2, create_using=None)
建立一个完全二分图
-- networkx.generators.bipartite.bipartite_configuration_model (aseq, bseq, create_using=None, seed=None)
根据两个度序列建立一个二分图
-- networkx.generators.bipartite.bipartite_random_regular_graph(d, n, create_using=None, seed=None)
建立一个随机的规则二分图
-- networkx.generators.bipartite.bipartite_preferential_attachment_graph(aseq, p, create_using=None, seed=None)
建立一个优先连接的二分图
-- networkx.generators.bipartite.bipartite_havel_hakimi_graph(aseq, bseq, create_using=None)
根据两个度序列建立一个Havel-Hakimi模式的二分图（下面两个模型类似，我没有接触过这个模型，不太理解具体含义）
-- networkx.generators.bipartite.bipartite_reverse_havel_hakimi_graph(aseq, bseq, create_using=None)
-- networkx.generators.bipartite.bipartite_alternating_havel_hakimi_graph(aseq, bseq, create_using=None)

二分性检验

```
nx.is_bipartite(B)
```

```
True
```

二分图投影

二分图可以通过向参与者节点投影或向项目节点投影转换为一个单分图（见下图），NetworkX提供的networkx.project(B, nodes)方法可以完成这一工作。它接受两个参数：一个是二分图B，另一个是投向的节点集合nodes。

对于节点集合的提取可以用networkx.bipartite_sets方法，它可以将一个二分图的两类节点提取为两个集合(X,Y)，其中X是项目节点，Y是参与者节点。下面是一段示例代码，演示上述两个函数的用法：

```
NSet = bipartite.sets(B)   #将二分图中的两类节点分别提取出来
Act = nx.project(B,NSet[0])     #向项目节点投影
Actor = nx.project(B,NSet[1])  #向参与者节点投影
```

```
Act.edges()  #输出 [(101, 102)]
Actor.edges()   #输出 [(201, 202), (201, 203), (202, 203), (203, 204)]
```

```
EdgeView([(201, 202), (201, 203), (202, 203), (203, 204)])
```

派系提取构造二分图

对于一个存在派系（Clique）的图，可以通过提取派系结构生成一个二分图。NetworkX提供的networkx.make_clique_bipartite方法可以从图中查找派系，然后将一个派系作为一个项目节点并和该派系中的节点建立连接，从而构造一个二分图。它是二分图向参与者节点投影的逆过程，下边是一段示例代码：

```
G = nx.make_clique_bipartite(Actor)
G.edges()
```

```
EdgeView([(201, -1), (202, -1), (203, -1), (203, -2), (204, -2)])
```

```

```