社会网络分析

周世祥

2020年5月21日

什么是社会网络

社会网络分析是社会学领域中一种比较成熟的定量分析方法，最初由社会学家根据数学方法、图论等发展起来，其本质是对网络社会、尤其是现代网络社会进行分析研究，因此，我们要先系统地探究诸如“什么是社会？”、“什么是网络社会？”、“什么是社会网络？”等一系列基本概念和基本问题。

从自然科学的角度来看，社会指的是由有一定联系、相互依存的个体组成的而又超乎于个人的、有机的整体。从社会科学的角度来看，社会被认为是人类所特有的(人类生活共同体)，由“个人”、“群体”、“群体中的社会关系总和(人与人、人与群体之间的生产、联系等)”这三者构成。虽然本书结合自然科学和社会科学展开交叉研究，但针对的研究对象仍然是人类社会，因此后面将提到的社会均等同于人类社会。

马克思认为，社会的本质是人和组织形式。人，确定了社会的规模和活动的状态。组织形式，决定了社会的性质，以及生产关系。应当注意到，社会并不简单地等同于人的群体：社会是人类相互有机联系，互相合作形成的群体；社会具有组织结构，社会成员之间具有紧密联系；社会具有相对集中统一并被成员认同的的价值取向和文化特征，社会成员有着生存和生产的职能和分工；社会具有对环境的适存和进化能力。

社会网络的构成

群体侧重于边界，社会则包含了边界以及边界中的结构。概括来说，社会具有6个特征：*(1)社会是有文化、有组织的系统。是由人群按照一定的模式组织起来的；*(2)生产活动是一切社会活动的基础，任何一个社会都必须进行生产；*(3)任何特定的历史时期，社会都是人类共同生活的最大社会群体；*(4)具体社会组织有明确的区域界限，存在于一定空间范围之内；*(5)社会有连续性和非连续性。任何一个具体社会都是从前人继承下来的一份遗产，同时*(6)社会有一套自我调节的机制，是一个具有主动性、创造性和改造能力的“活的有机体”，能够主动地调整自身与环境的关系，创造自身生存与发展的条件。

前面提到，社会个体成员之间由互动而形成相对稳定的关系体系， 在这种关系体系的交织下，整个社会的拓扑呈现就如同一张网络。网络中的节点是社会的个体或组织；网络中的边线是社会中的关系与联系，社会关系包括朋友关系、同学关系、生意伙伴关系、种族信仰关系等。这就是“社会网络”/“网络社会”的名称由来。

社会网络(social network)是一种基于“网络”(节点之间的相互连接)而非“群体”(明确的边界和秩序)的社会组织形式，也是西方社会学从 1960 年代兴起的一种分析视角。随着工业化、城市化的进行和新的通讯技术的兴起，社会呈现越来越网络化的趋势，发生“社会网络革命”(social network revolution)，与移动革命(mobile revolution)、互联网革命(internet revolution)并列为新时期影响人类社会的三大革命。尤其在计算机网络时代的今天，人类社会更加网络化而趋近于“网络社会” ，网络社会的社会网络分析也就越来越重要

社会网络分析的是对网络社会的结构(网络中的边)进行量化，将人/群体在社会环境中的相互作用表达为基于关系的一种模式或规则，从而发掘出蕴含/隐藏在社会网络中的某种规律，提高政府/机构/组织的社会管理/决策水平。如今社会网络分析涉及多个学科和研究领域，例如：数据挖掘、知识管理、数据可视化、统计分析、社会资本、小世界理论、信息传播等。

正如西方的一句著名格言：“重要的不在于你懂得什么(whatyouknow)，而在于你认识谁(whoyouknow)。”社会网络分析有别于传统的统计分析和数据处理方法，其更加关注的是社会网络中一组个体之间关系，这组个体可以是人、社区、群体、组织、国家等，从关系中提取出来数据和模式，并反映/预测社会现象是社会网络分析的焦点。

显然，“网络社会”之“网络”，在互联网或电脑网络出现之前就存在；“网络”并非专指互联网(Internet)或电脑网络(Computer network)，而是“一组相互连接的节点(nodes)”。至于具体的节点是什么，则根据我们所谈的具体网络种类而定。在现实生活中，从个人、家庭、组织乃至国家都存在于与其他单位的关系网络中作为节点；人类社会

正是由各种具体社会网络组成，社会资源则在这些网络中流动，社会运行就在于这些网络的稳定和其中资源的顺畅交流。如图8-5所示，这种传统意义上的“网络社会”早就存在，虽然它的形式在漫长的人类历史中变化缓慢。

随着20 世纪90年代信息技术革命的掀起，产生了不同于以往任何网络的崭新的“网络”——基于计算机网络技术的“电脑网络”或“信息网络”，这开始为网络形式的大变革奠定了物质基础，也崛起了现代意义上的“网络社会”。高度发展的信息网络，一方面通过现实社会的投射，构成了自己虚拟的“网络社会”(Cyber society)；另一方面通过信息网络的渗透，融合了各种已存的社会实体网络，使“网络社会”(Network society)成为整个现实社会的结构形态(使得传统的网络社会更加网络化)，我们称这个“网络社会”(Network society)为“现代网络社会”。

显然，“虚拟网络社会”(Cyber society)是“现代网络社会”(Network society)的基础，而又被包容在后者之中。“现代网络社会”是以Internet为主的信息网络与实体网络高度整合的结果，也是虚拟“网络社会”和现实“网络社会”高度整合的结果。在目前的学术界中，“虚拟网络社会”的研究和“现代网络社会”的研究同样都是时下的热门话题。有学者主张将这两者都统称为“现代网络社会”， 针对“现代网络社会”的社会网络分析——基于新媒体数据挖掘和R开源软件平台。

网络关系的描述

概括来说，社会网络分析中的关系网络可分为组织关系网络、情感关系网络、咨询关系网络三大类。而在离散数学/图论中，“关系”就是一种二元之间的单向/双向属性；与此类似地，主流社会科学所关注的关系是一对“行动者”之间的二元属性。社会活动中存在着不同的二元属性关系有：

血缘关系：是谁的兄弟，是谁的父亲，婚姻关系等

社会角色：是谁的领导，是谁的教师，是谁的朋友等

情感关系：喜欢谁，尊敬谁，恨谁等；

认知关系：知道谁，与谁看起来相似等；

行动关系：同谁谈话，一同吃饭，进攻谁，传递信息给谁，从谁接受信息等；

流动关系：汽车流量，信息流量，通信流量等

距离关系：两地距离；

相似关系：相关系数度量；

共同发生：同一个俱乐部，有相同颜色头发等；

自20世纪90年代以来，随着计算机技术的不断发展和网络分析理论研究的深入，社会网络分析的模型得到了进一步的改进，社会网络分析跨越了传统的学科界限从而越来越多的运用到了各个领域。社会网络分析逐渐成为了一种跨学科的研究方法。

社会网络分析的几个重要指标

利用计算机社会网络研究软件绘制网络图，根据需求计算出各部分指标，利用这些内容对所要分析的对象进行分析。例如UCINET、NodeXL、R、python可以进行社会网络分析的指标都很多，比较常见的就有网络密度、中心性、凝聚子群等指标。

社交网络中的几个重要度量：

中心度(Degree)：表示节点的链接数量。如果网络中的链接都是强关系，那么度中心性高的人就比度中心性低的人受欢迎。图8-9是一个训练班的同学之间的关系(下面就以代号TC表示)，各向量指标数如图8-8所示，Bill度中心性是6，Tom和Mark的是1，说明Bill最受欢迎，而Tom和Mark则比较边缘化。在有向图中，度中心性分为出度中心性(Out-Degree)和入度中心性(In-Degree)。A的出度中心性是从A节点指向其它节点的链接数，A的入度中心性是指向A节点的链接数。

中介中心性(Betweenness Centrality)：在路径上能够到达其它节点的度量。在TC中，Bill的中介中心性是17，Joseph是14，远高于其它节点，因为如果没有Joseph，Willian和Tom就会与其他人断开关系，同样，Mark必须通过Bill与其他人建立关系。而James、Mark、Henry和Tom的中介中心性为0，因为他们不在其他关系的路径上。

接近中心性(Closeness Centrality)：有能力在最短路径到达其它节点的度量。接近中心性的值越小说明节点更容易到达其它节点，即平均路径最短。

特征向量中心性(Eigenvector Centrality)：不仅考虑链接总数，还考虑与谁连接。即“把那些与特定行动者相联结的其他行动者的中心性考虑进来而量度一个行动者中心性指标”。

NodeXL是一个功能强大且易于使用的交互式网络可视化和分析工具，它以MS Excel（Excel 2007或者Excel 2010）模板的形式，利用MS Excel作为数据展示和分析平台。可以定制图像外观、无损缩放、移动图像，动态过滤顶点和边，提供多种布局方式，查找群和相关边，支持多种数据格式输入和输出。

社交软件(如QQ，微信)可以完整地表现人们的生活，其实就是无形中的庞大社会网络。人们用不同的方式与他人互动，并且这些信息都可以在社交软件中抓取到。挖掘某个站点的有用信息可以帮助一些团体增加竞争力。R提供的“iGraph”软件包提供了一些非常有效的挖掘功能(通常需要RCrul、tm、stringr、network等众多扩展包的配合)。

```
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

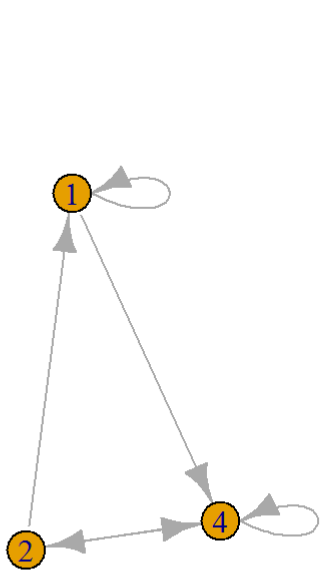
```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
# Create a directed graph
g <- graph(c(1,2, 1,3, 2,4, 1,4), directed=T)
g
```

```
## IGRAPH 64b5a3e D--- 4 4 --
## + edges from 64b5a3e:
## [1] 1->2 1->3 2->4 1->4
```

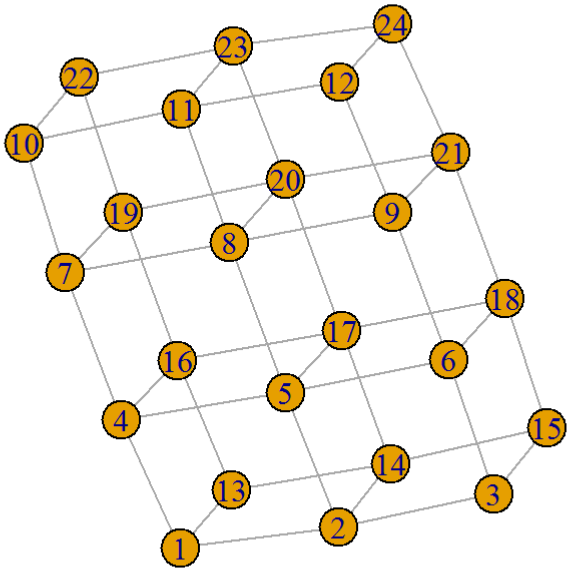
```
library("igraph")
g<- graph(c(1,2, 1,3, 2,4, 1,4),directed=T)
m<-matrix(runif(4*4),nrow=4)
g<- graph.adjacency(m>0.5)
plot(g,layout=layout.fruchterman.reingold)
```



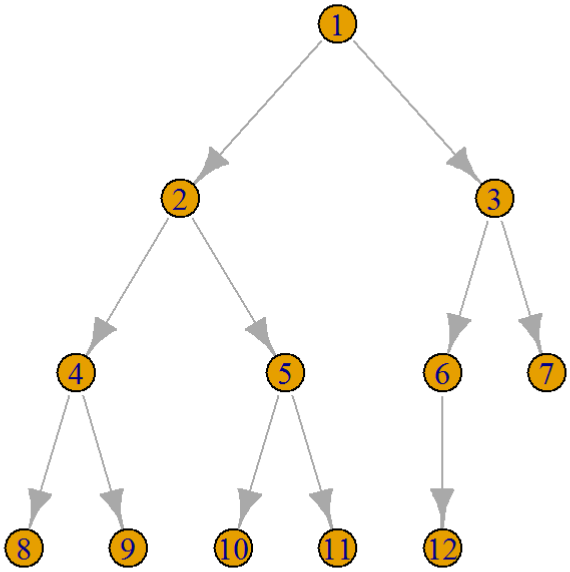

```
g1<- graph.full(4)
g2<-graph.ring(3)

g<-g1 %du% g2
graph.difference(g,graph(c(1,2,1,3),directed = F))
```

```
## IGRAPH 64ddec6 U--- 7 7 --
## + attr: name_1 (g/c), name_2 (g/c), loops (g/l), mutual (g/l), circular
## | (g/l)
## + edges from 64ddec6:
## [1] 1--4 2--4 2--3 3--4 5--7 5--6 6--7
```

```
g1<-graph.lattice(c(3,4,2))

g2=graph.tree(12,children = 2)
plot(g1,layout=layout.fruchterman.reingold)
```

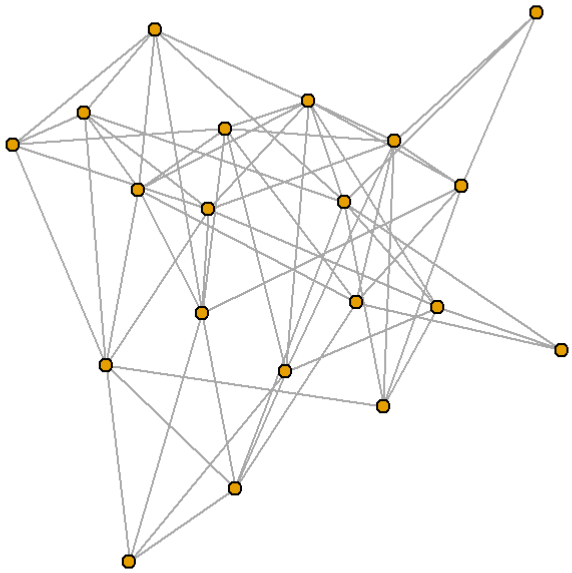



```
plot(g2,layout=layout.reingold.tilford)
```



Graph还提供了另外两种图表生成的机制。“随机图表”可以在任意两个节点之间进行连线。而“优先连接”会给已经拥有较大度数的节点再增加连线（也就是多者更多）。

```
# Generate random graph, fixed probability
 g <- erdos.renyi.game(20, 0.3)
plot(g, layout=layout.fruchterman.reingold, vertex.label=NA, vertex.size=5)
```



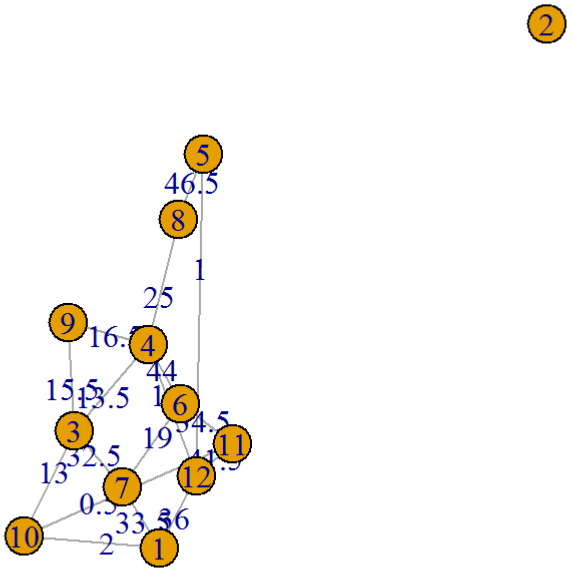
```
# Generate random graph, fixed number of arcs
g <- erdos.renyi.game(20, 15, type='gnm')

# Generate preferential attachment graph
g <- barabasi.game(60, power=1, zero.appeal=1.3)
```

简单图表算法

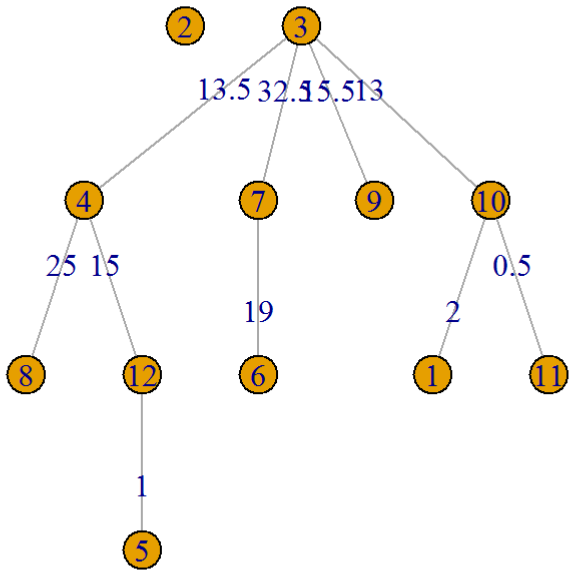
这一节会介绍如何使用iGraph来实现一些简单的图表算法。

最小生成树算法可以在图表里连接所有的节点，并使所有的连线权重最小。

```
# Create the graph and assign random edge weights
  g <- erdos.renyi.game(12, 0.35)
  E(g)$weight <- round(runif(length(E(g))),2) * 50
  plot(g, layout=layout.fruchterman.reingold, edge.label=E(g)$weight)
```




```
# Compute the minimum spanning tree
   mst <- minimum.spanning.tree(g)
  plot(mst, layout=layout.reingold.tilford,  edge.label=E(mst)$weight)
```



连通分支算法可以找到会连通其他节点的连接，也就是说，两个节点之间的路径会穿过其他节点。需要注意的是，在无向图里连通是要对称的，在有向图（节点A指向节点B，但节点B不指向节点A的图表）里不是必须的。因此在有向图中存在一种连接的概念叫做“强”，也就是只有两个节点都分别指向对方才意味着它们是连通的。“弱”的连接意味着它们不是连通的。

```
g <- graph(c(0, 1, 1, 2, 2, 0, 1, 3, 3, 4, 4, 5, 5, 3, 4, 6, 6, 7, 7, 8, 8, 6, 9, 10, 10, 11, 11, 9)+1)
# Nodes reachable from node4
  subcomponent(g, 4, mode="out")
```

```
## + 6/12 vertices, from 654efdb:
## [1] 4 5 6 7 8 9
```

```
# Nodes who can reach node4
  subcomponent(g, 4, mode="in")
```

```
## + 6/12 vertices, from 654efdb:
## [1] 4 2 6 1 5 3
```

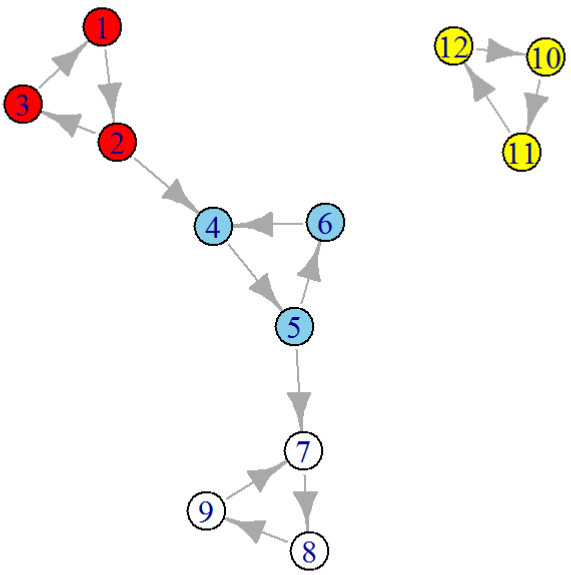
```
  clusters(g, mode="weak")
```

```
## $membership
##  [1] 1 1 1 1 1 1 1 1 1 2 2 2
##
## $csize
## [1] 9 3
##
## $no
## [1] 2
```

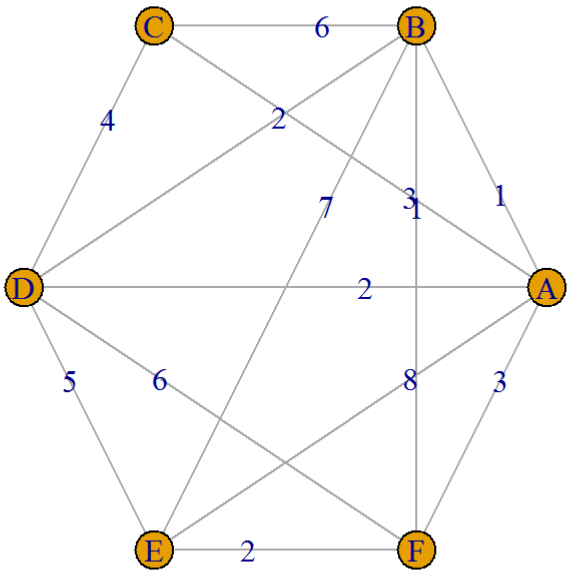
```
  myc <- clusters(g, mode="strong")
  myc
```

```
## $membership
##  [1] 2 2 2 3 3 3 4 4 4 1 1 1
##
## $csize
## [1] 3 3 3 3
##
## $no
## [1] 4
```

```
  mycolor <- c('green', 'yellow', 'red', 'skyblue')
  V(g)$color <- mycolor[myc$membership + 1]
  plot(g, layout=layout.fruchterman.reingold)
```



```
library(igraph)

w <- c(1,
       3, 6,
       2, 2, 4,
       8, 7, 0, 5,
       3, 1, 0, 6, 2)
W <- matrix(0, nc = 6, nr = 6)#根据邻接矩阵，作图
for (i in 2:6)
   for (j in 1:(i-1))
      W[i,j] <- w[(i-1)*(i-2)/2+j]
colnames(W) <- LETTERS[1:6]
g8 <- graph.adjacency(W, mode="lower", weighted=T)
#lower表示无向图，只读取对角线下部

label<-c(1, 3, 6, 2, 2, 4, 8, 7, 5, 3, 1, 6, 2)
plot(g8, layout = layout.circle, edge.label = label)
```



求从起点到终点的最短路径

```
Vnames = c("u0", "v1", "v2", "v3", "v4", "v5",
           "v6", "v7", "v8", "v9", "v10")
d <- data.frame(
    from = c("u0", "u0", "u0", "v1", "v1", "v2", "v2",
       "v2", "v2", "v3", "v4", "v4", "v4", "v5", "v5",
       "v6", "v6", "v7", "v7", "v8", "v8", "v9"),
    to   = c("v1", "v2", "v3", "v2", "v4", "v3", "v4",
       "v5", "v6", "v6", "v5", "v7", "v8", "v6", "v8",
       "v8", "v9", "v8", "v10","v9", "v10","v10"),
    weight = c(2,8,1,6,1,7,5,1,2,9,3,2,9,4,6,3,1,7,9,1,2,4)
)
g <- graph.data.frame(d, directed = F, vertices = Vnames)
shortest.paths(g, v = V(g)['u0'], to = V(g)['v10'])
```

```
##    v10
## u0  13
```

```
gs <- get.shortest.paths(g, from = V(g)['u0'], to = V(g)['v10'])
V(g)[gs$vpath[[1]]]
```

```
## + 9/11 vertices, named, from 6583c95:
## [1] u0  v1  v4  v5  v2  v6  v9  v8  v10
```

```
#从顶点u0到顶点v10的最短路
```

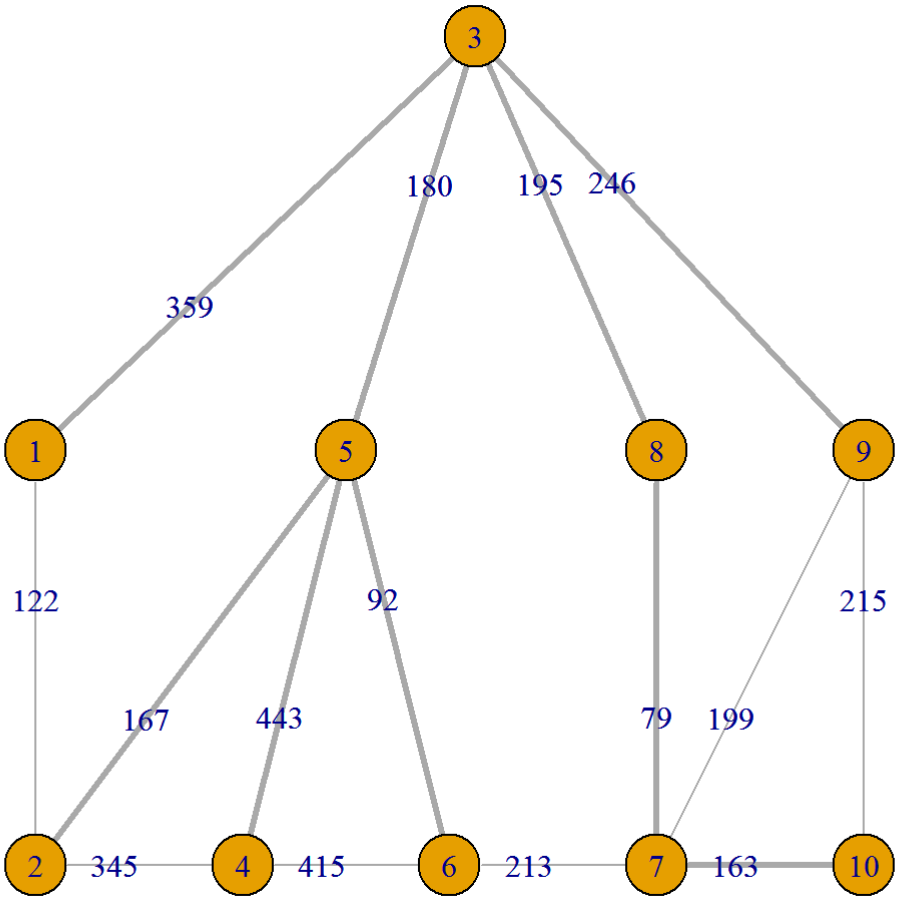
一个模型

某省共有10个大中城市，各城市之间的高速公路交通示意图如图，在城市3建有该省物资集散中心，物资将从这里出发，运往其他城市，请计算从城市3到各城市之间的最短距离，并设计相应的行车路线。

```
library(igraph)

E <- c(1,2, 1,3, 2,4, 2,5, 3,5, 3,8, 3,9, 4,5, 4,6,
       5,6, 6,7, 7,8, 7,9, 7,10, 9,10)
W <- c(122, 359, 345, 167, 180, 195, 246, 443, 415,
        92, 213,  79, 199, 163, 215)
g <- graph.empty(directed=F)+vertices(1:10)+edges(E)

sp <- shortest.paths(g, v = 3, weights = W)
gs <- get.shortest.paths(g, from = 3, weights = W)

for (i in 1:10)
    if (i != 3)
        print(c(i, sp[i], gs$vpath[[i]]) )
```

```
##           3   1
##   1 359   3   1
##           3   5   2
##   2 347   3   5   2
##           3   5   4
##   4 623   3   5   4
##           3   5
##   5 180   3   5
##           3   5   6
##   6 272   3   5   6
##           3   8   7
##   7 274   3   8   7
##           3   8
##   8 195   3   8
##           3   9
##   9 246   3   9
##           3   8   7  10
##  10 437   3   8   7  10
```

```
E(g)$label<-W
par(mai=c(0,0,0,0))
plot(g,layout=layout.reingold.tilford)

E(g, P=c(3,1, 3,5, 5,2, 5,4, 5,6, 3,8, 8,7, 3,9, 7,10))$width<-3
plot(g,layout=layout.reingold.tilford(g), add=T)
```



最短路径算法是最普遍的算法，它能找到节点A和节点B之间最短的路径。在iGraph里，如果图表是未加权的（也就是权重为1的）而且在权重为正时使用了迪杰斯特拉算法，会使用“breath-first search”算法。要是连线的权重是负数，则会使用Bellman-ford算法。

```
# g <- erdos.renyi.game(12, 0.25)
# plot(g, layout=layout.fruchterman.reingold)
# pa <- get.shortest.paths(g, 5, 9)[[1]]
# pa
#
# V(g)[pa]$color <- 'green'
# E(g)$color <- 'grey'
# E(g, path=pa)$color <- 'red'
# E(g, path=pa)$width <- 3
# plot(g, layout=layout.fruchterman.reingold)
```

github社交编程平台

GitHub于2008年4月10日正式上线，GitHib原本是一个面向开源及私有软件项目的托管平台，因为只支持git 作为唯一的版本库格式进行托管，故名gitHub。随着平台的不断发展与壮大，除了git代码仓库托管及基本的 Web 管理界面以外，Github又陆续提供了订阅、讨论组、文本渲染、在线文件编辑器、协作图谱(报表)、代码片段分享(Gist)等功能，并逐渐演化为极其重要的社交编程平台。

GitHub的核心思想是以开源分布式版本控制系统Git为开发者提供有益的代码托管解决方案，用来建立和维护开源软件项目，能够让程序员分享和共同协作开源项目的开发。目前，其注册用户已经超过350万，托管版本数量已超过600万，其中不乏知名开源项目 Ruby on Rails、jQuery、python 等。不同于CVS或Subversion，Git本身并不需要像传统方式那样复制代码库。所有的副本都是工作副本，开发者可以将工作时的副本提交至本地的动态软件仓库，而不需要连接中央服务器。

和他社交平台一样，GitHub提供了供开发者使用的网站。提供了开发者注册、文档下载等支持。

简单来说，GitHub的API就是让我们能够通过编程语言调用的语句，调用API可以实现请求各种信息。例如在浏览器的地址栏中访问如下地址,就是对GitHub API的调用： https://api.github.com/search/repositories?q=language:r&sort=stars (https://api.github.com/search/repositories?q=language:r&sort=stars)

和其他社交网络软件一样，GitHub API是在建立在HTTP协议之上的，并且可以使用任何能发出HTTP请求的编程语言来访问，包括终端的命令行工具。

事实上，上述直接调用GitHub Api的方式是有所限制的。 在浏览器中输入https://api.github.com/users/pywxf 并打开，收到了JSON格式的响应数据。可以看到，凡是属于没认证的客户端每小时仅可以制造60个http请求，如果想做到更多，则必须进行身份认证。不仅如此，使用GitHub API做任何有趣一点的事情都会要求认证(未认证的请求，功能也受到限制)。因此，我们应该在GitHub上对我们的API访问进行身份认证。

GitHub API最简单的认证方式是使用你自己的 GitHub 用户名和密码来通过基础认证，认证后重新访问(https://github.com/settings/developers (https://github.com/settings/developers)， 实现起来其实非常简单)会看到你的速度限制会升到每小时5000个请求，这个会在X-RateLimit-Limit头部信息中标示。基本认证虽然方便，但是并不理想，因为显然你不应该将你的GitHub用户名和密码共享给任何人。和其他社交网络一样，GitHub实现了OAuth(Open Authorization)，OAuth 用令牌(token)替代了用户名和密码. 试着查阅资料并剖析OAUTH的工作原理

如何获取GitHub API的资料 推荐初学者学习极客学院推出的简化教程 ，网址为：http://wiki.jikexueyuan.com/project/github-developer-guides/ (http://wiki.jikexueyuan.com/project/github-developer-guides/)

参考文献

王小峰等，《新媒体数据挖掘，基于R语言》，清华大学出版社，2017年10月。

薛毅，《数学建模：基于R》，机械工业出版社，2017年7月。

https://www.ituring.com.cn/article/1762 (https://www.ituring.com.cn/article/1762)