图论算法的程序实现

马仁杰

（清华大学 电子工程系 2017080061）

前言：图论算法能用来解决各种问题，并且在计算机中有较多应用。因此，如何用计算机程序来实现离散数学中的图论算法是一个值得探究的课题。在本专题研究中，分别实现了 Kruskal 算法和 Dijkstra 算法，并“顺便”编写了一个处理图、树和二部图的软件库，可以用于后续相关问题的研究。本次研究的一个目标是用图形编程将算法的过程及结果可视化，这样不仅能够清晰地显示出算法运行的结果，同时，能够看到算法一步一步地进行，能让人直观地理解算法是如何运行的。

(一) Kruskal 算法

1. 算法介绍
   a) 最小生成树问题

      Kruskal 算法解决的是图论中的最小生成树问题（Minimum Spanning Tree Problem）。最小生成树问题即在连同加权图中（Connected Weighted Graph），找到权重最小的生成树（Minimum Spanning Tree）

   b) 算法流程

      1. 建立连通图 G(V, E)，生成树 F，
      2. 将 E 中的边按权重从小到大顺序排列
      3. 选择最短的边，假如与目前的生成树形成闭的路（cycle）则去掉它，否则加到 F 中
      4. 重复步骤 3，直到 F 中有 n-1 条边

   c) 算法应用

      Kruskal 算法最经典的应用是供电电缆的铺设问题。电缆只能按照固定的线路铺设，由于不同线路的长度及铺设深度不同，成本也各不相同。通过 Kruskal 算法可以在已有的线路中选出最佳铺设方案（最小生成树），以最低成本将全部的房子接入线路。

      此外，Kruskal 算法在以下领域中也有应用：

      - 分类学[1]
      - 电路设计[2]
      - 聚类分析[3]
      - 分析生态毒理学信息[4]

2. 程序编写
   a) 目的、预期及语言选择

在本次研究中的主要目的是演示算法的过程。我想通过图形编程表示出算法运行的步骤和结果。因此，我选择 Processing 作为编程语言。Processing 是基于 Java 的开源语言和集成开发环境（IDE），旨在简化图形编程，鼓励视觉艺术、学生等不同人群尝试编程。Processing 的内置函数极大地简化了线条、圆圈、点等基本图形的绘制，为图论算法的视觉呈现提供了基础。同时由于 Processing 基于 Java，因此确保了语言在运算、逻辑和数据处理上的功能性。

b) 离散数学基础

在开始构建算法的高级逻辑前，我需要搭建一些基本的功能和函数，使一些图论的基本概念和操作能够通过代码来表示。在此，面向对象编程（OOP）起到很大作用。在面向对象编程中，可以定义一个类型（例如端点），这个类型可以有不同的成员变量（例如端点的度）和成员函数（又称方法），例如在屏幕上显示端点是一个函数。面向对象的一个有点是可以将数据和代码进行封装（encapsulation），也就是把这些变量和函数变为一个单独对象的特性。这样就极大地方便主函数逻辑的设计。例如主函数要在图中若要添加一个端点，无需知道在屏幕上显示该端点的过程。

因此，在设计算法本身的逻辑前，我编写了一个库（library），该库定义了图、树及端点、边的变量，例如点的度、边的端点、图的点集合和边集合等等，以及相关的函数，例如通过 Püfer 码还原树，在屏幕上打印边，等等。这样子，在设计程序逻辑时，可以直接对图论的元素进行操作。关于库的编写，在下一段会有详细介绍

c) 库的编写

i. 端点 Node

在图论中，边可以看作是点集的子集，图可以看作是点集合和边集合构成的对象。因此，端点是图的最基本构成。因此需要定义出端点这个类型，其中这个类型的变量包含两类，一类是端点的在图论的定义中的性质，例如度和邻点，一类是计算机在处理端点时用到的变量，例如横纵坐标（在图论中端点的位置是无关的）。

Class Node

变量（函数）名	类型	说明
nodeX, nodeY	int	横坐标，纵坐标
uid	int	端点编号
nodeColor	Color	显示颜色
degree	int	端点的度
neighbors	arrayList	端点的邻点的集合
Node(int nodeX, int nodeY, int uid)	void	用坐标和编号初始化
display()	void	在屏幕上显示端点

ii. 边 Edge

Class Edge

int edgeX1, edgeY1, edgeX2, edgeY2	int	两端点的横纵坐标，初始化时自动从端点集成
edgeColor	color	显示颜色
weight	int	边的权重，默认用长度代表权重
nodes	Node[]	两个端点
Edge(Node node1, Node node2)	void	用两个端点初始化
display()	void	显示边

iii. 图 Graph

Class Graph

nodes	ArrayList	点集合
edges	ArrayList	边集合
node_count	int	点数
edge_count	int	边数
gather_code	int[][]	Father code*
Graph(int node_count, edge_count)	void	用点数和边数初始化
draw_from_father_code()	void	通过 father code 得出端点和边的关系
draw_nodes()	void	画出端点
draw_edges()	void	画出边
initialize_nodes()	void	通过边集合确立每个端点的度和邻点，并更新每个点的相关变量

*此处的 father code 不是严格意义上的 father code，真正的 father code 能够体现端点之间的父子关系，该处的 father code 仅需体现出每条边两端点即可（例：{0, 3},{1.2}代表编

号为 0 的端点与编号为 1 的端点之间有一条边，编号为 3 的端点与编号为 2 的端点之间有一条边）

iv. 树 Tree

Class Tree extends Graph

prufer_code	int[]	树的 Püfer 码
construct_from_prufer_code()	void	通过 Püfer 码得出 father_code，进而构造出树

v. 二部图 Bipartite Graph

Class Bipartite extends Graph

spacing	int	点间距
dscolumn_distance	int	列间距
startX, startY	int	第一个点（左上）的横纵坐标
Bipartite(int node_count, int edge_count, int startX, int startY, int spacing, int column_distance)	void	用点数、边数、起始坐标和间距初始化二部图

d) 程序逻辑

i. 主函数逻辑：

1. 初始化完全图 graph，树 tree
2. 用嵌套循环生成一个完全图的 father code，并将其赋予 graph
3. 运行 draw_from_father_code()，通过 father code 的到端点的度、邻点，以及每条边的两端点和权重（长度）
4. **将 graph 的边按照权重进行排序（冒泡排序）**
5. 将 tree 的每一个端点的坐标设为对应 graph 每个端点的坐标（在初始化 tree 时会随机生成每个端点的坐标，为了方便观察在此处令最小生成树的每个端点与完全图的每个端点重合
6. **设定两个计数器 edge_counter 和 counter_graph，初始值为 0**

（以下部分循环执行）

7. **若 edge_counter 小于最小生成树的边数（端点数-1）并且 counter_graph 小于完全**

图的边数（端点数的组合数）：

a) edge 为完全图的边按权重排序后序号为 counter_graph 的边

b) 若在 tree 中（注意不是 graph！）edge 的两端点不存在一条路（path）：

i. 令新边 edge_temp 等于 edge

ii. 令 edge_temp 的颜色为绿色

iii. 对 tree 中的全部端点，若存在端点等于 edge_temp 的一个端点的编号（uid）相等，则令 edge_temp 的端点等于 tree 中该端点，并且该端点的度+1*

iv. 将 edge_temp 加到 tree 中

v. 在 tree 调用 initialize_edges()，更新每个端点的度和邻点集合**

vi. 显示 edge_temp

c) 若 tree 中 edge 两端点存在一条 path：

i. 令 edge 的颜色为红色

ii. 显示 edge

8. 否则（算法完成）：

a) 令红色的边变为白色（擦去）

b) 从新显示 tree 的边和端点

9. counter_graph+1

*由于 edge 是 graph 的边，因此 edge 的端点是 graph 的点而不是 tree 的端点（虽然 tree 的端点与 graph 的端点完全相同），因此该步骤的目的是将 edge_temp 的端点全部变为 tree 中的端点。

**由于在判断两点间是否存在 path 的函数（以下有具体说明）会用到 tree 的端点的邻点和度，因此在 tree 中每添加一条边都要更新邻点和度

ii. 判断两端点之间是否存在一条 path 的函数流程如下：

1. 给定两端点 start, end

2. 建立新的点集合 list（类型为 ArrayList<Node>），并将 start 加到 list 中

3. 当 list 中不存在 end 时，

a) 新建点的集合 temp，并将 list 集合中每个点的所有邻点加入到 temp 中

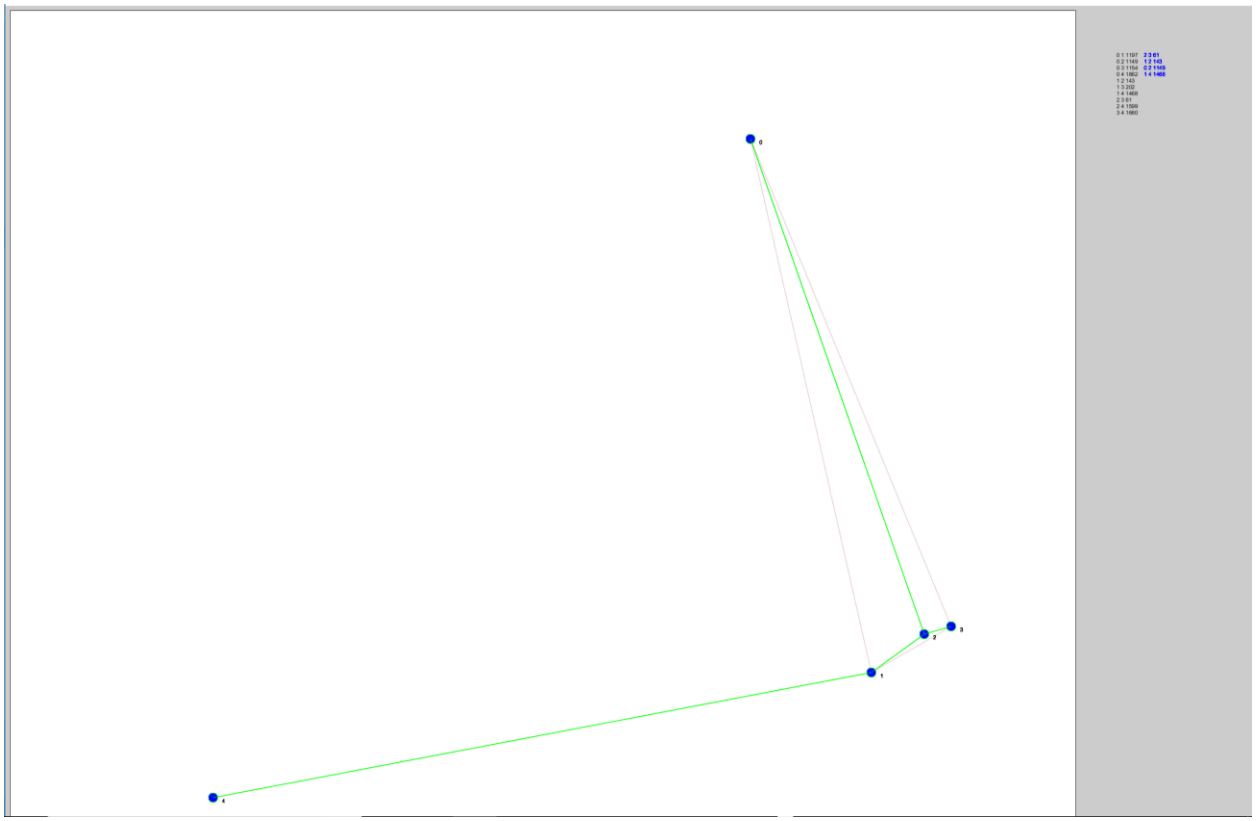
b) 若 temp 为空集，则 start 和 end 之间不存在一条 path，函数终止

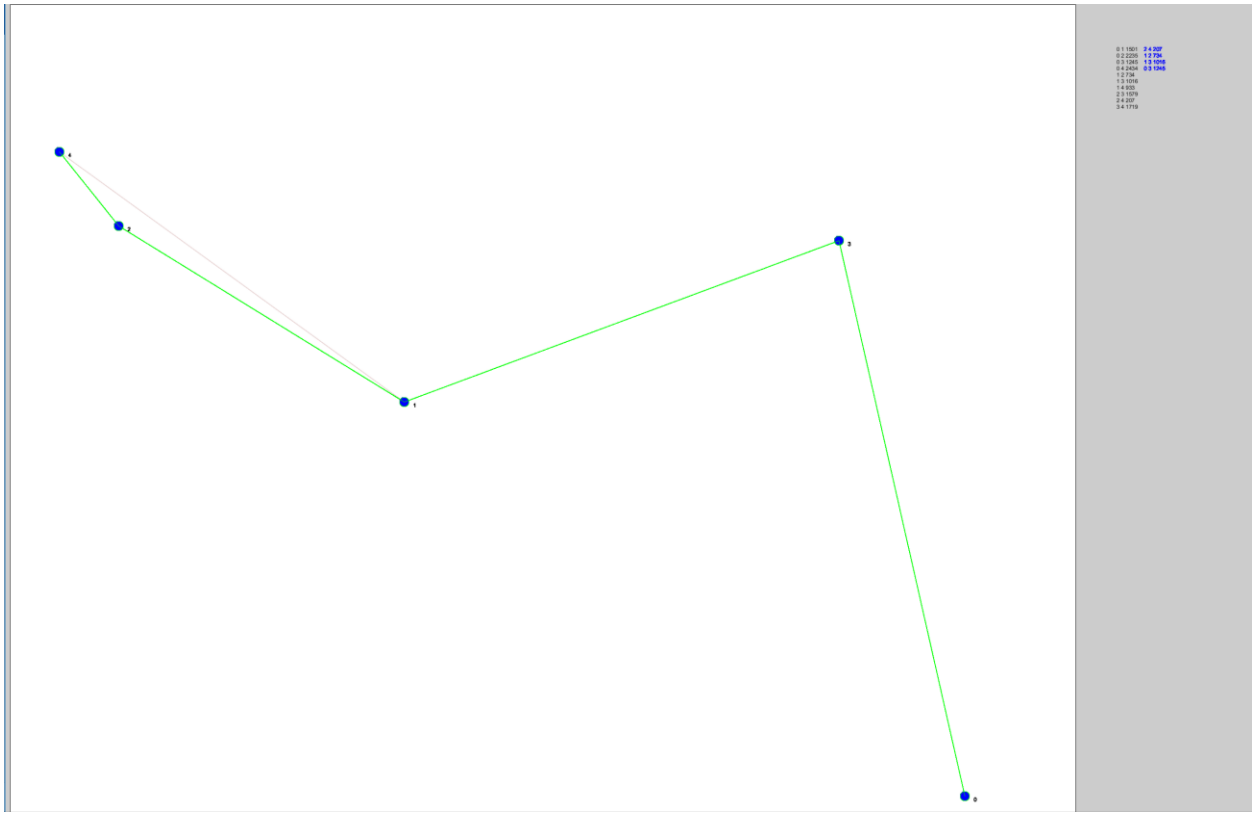
c) 否则将 temp 并入 list 中

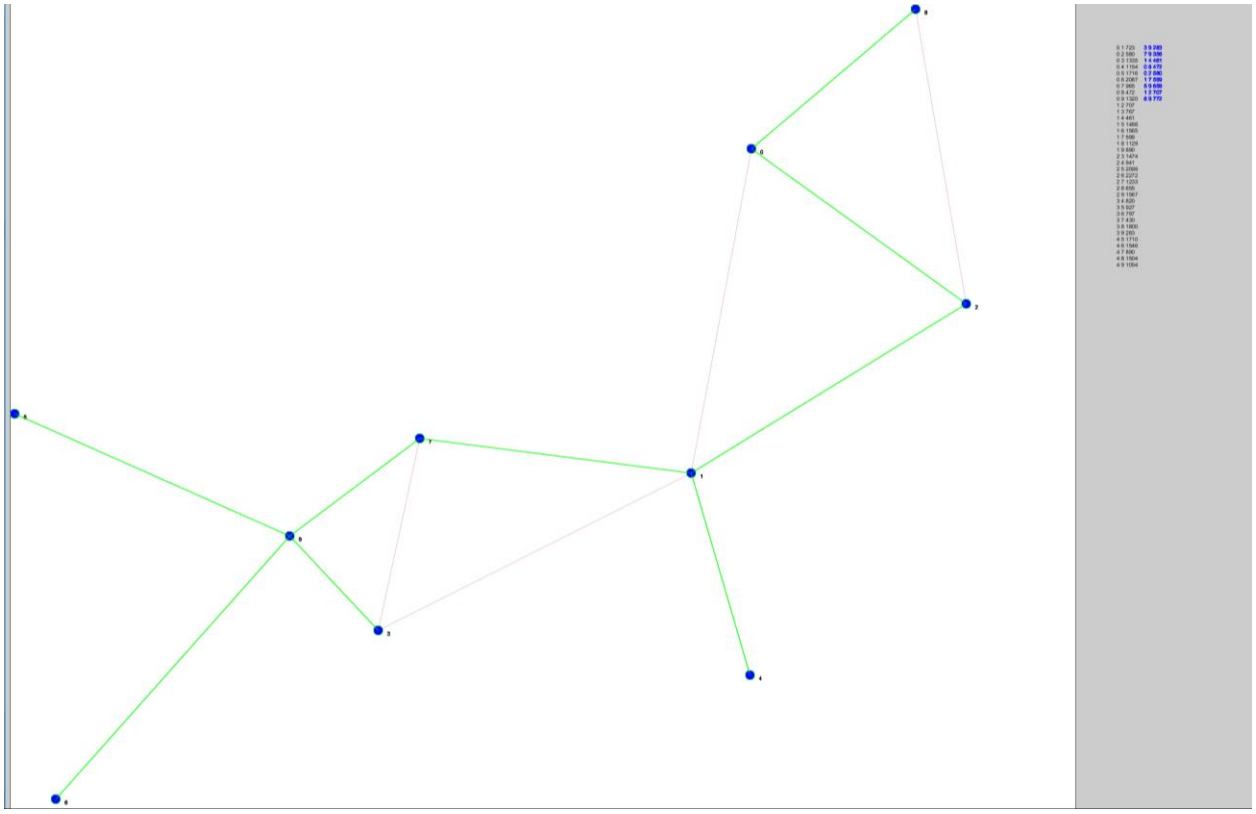
4. （若 list 出现了 end）start 和 end 之间存在一条 path，函数终止

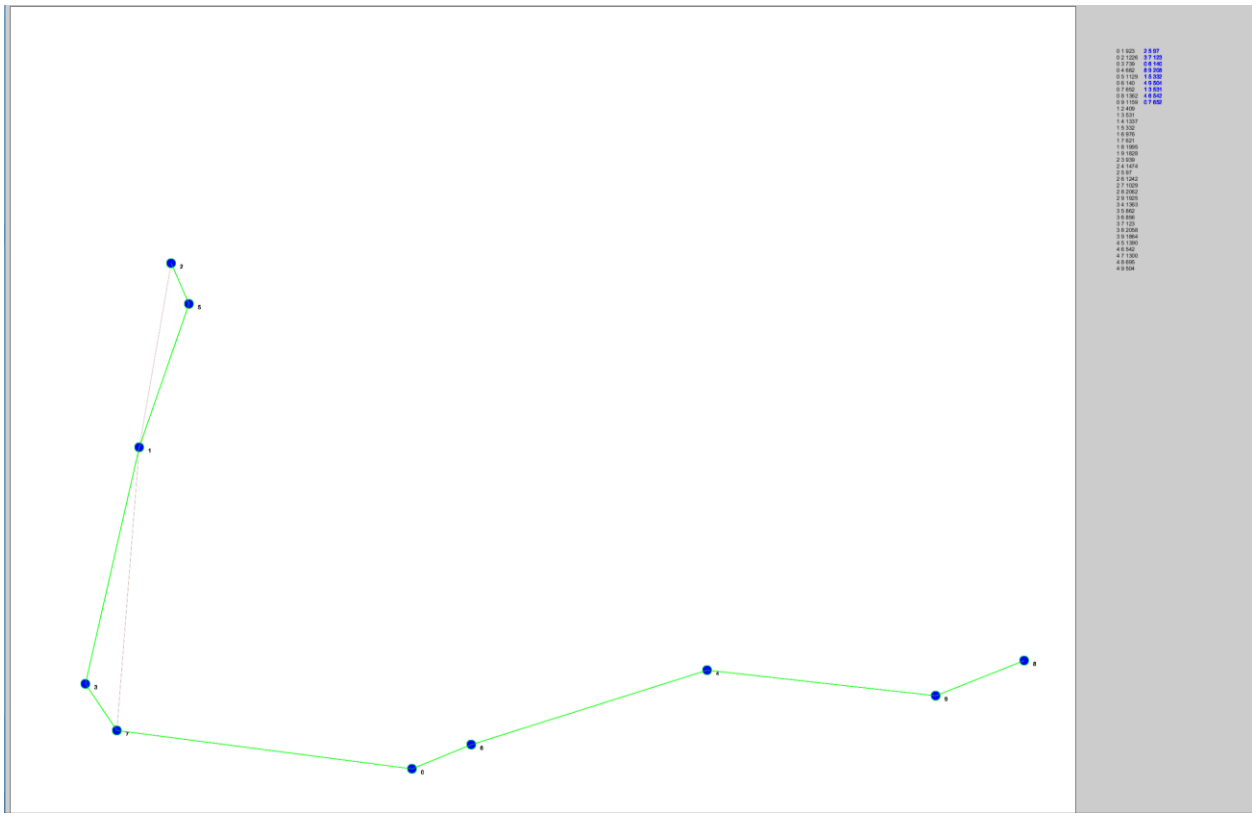
3. 程序测试

测试的两大指标为正确性（准确性）及执行速度。在正确性的测试中需要用较少的端点数和较低的帧率进行测试。这样可以比较容易观察出过程并判断结果的正确性。因此分别选用 5 个端点和 10 个端点以 2 帧/秒的速度进行两次，。运行结果截图如下：


0 1 1197 3 3 61
0 2 1149 1 2 143
0 3 1154 0 2 1149
0 4 1852 1 4 1468
1 2 143
1 3 202
1 4 1468
2 3 61
2 4 1599
3 4 1650
0
1
2
3
4


0 1 1501 2 4 207
0 2 2235 1 2 734
0 3 1343 1 3 1016
0 4 2434 0 3 1345
1 2 734
1 3 1016
1 4 933
2 3 1579
2 4 207
3 4 1719
4
2
3
1
0


0 1 725 3 9 283
0 2 580 7 9 386
0 3 1335 1 4 481
0 4 1164 0 8 472
0 5 1716 0 2 580
0 6 2067 1 7 699
0 7 1086 1 9 890
0 8 472 1 5 707
0 9 1328 6 9 772
1 2 707
1 3 787
1 4 481
1 5 1466
1 6 1905
1 7 699
1 8 1129
1 9 890
2 3 1471
2 4 1164
2 5 2089
2 6 2272
2 7 1233
2 8 655
2 9 1167
3 4 820
3 5 927
3 6 787
3 7 430
3 8 1800
3 9 1161
4 5 1110
4 6 1946
4 7 890
4 8 1924
4 9 1054
8
0
2
5
7
1
9
3
4
6



可观察到绿线为最小生成树，红线形成闭的路。右侧黑字分别为每条边的两端点及长度，蓝色的字为最小生成树的边的端点及长度。经多次验证确定算法无误。

（二）Dijkstra 算法

1. 算法介绍

   a) 解决问题

   Dijkstra 算法用于解决最短路径的问题，即在加权图中找出给定两点的最短路径。

   b) 算法流程[5]

   1. 定义起始点 I，终止点 E，为每个点定义一个距离值，令 I 的距离值为 0，其他点的距离值为正无穷
   2. 令当前点 V 为 I，其他点为未经过的点
   3. 对于当前点 V，考虑 V 的全部邻点并算出他们与 V 的距离（或者他们与 V 分别形成的边的权重），并 V 的距离值相加（例：若 V 为起始点 I，I 与邻点 B 的距离为 6，则算出 6 与 I 的距离值 0 的和）并与邻点的距离值进行比较，若该值小于邻点的距离值则令距离值等于算出的这个值
   4. 处理完 V 的全部邻点后，将 V 从未经过点的集合中去掉。
   5. 选择未经过点中距离值最小的点作为当前点 V，重复 3，4，直到经过终止点 E（或

没有未经过的点）。

很显然，以上步骤得到的是最短路径的距离，无法直接得到路径本身；若想得到具体的路径，还需要做一点改进：每次改变一个邻点的距离值时，记下对应的当前点 V（我们成为父节点 father node）。算法运行后，将终止点与其父节点相连，父节点与其父节点相连，一次下去，连到起点为止。

c) 算法应用

Dijkstra 算法最常见的应用是在手机导航软件中，不同的公路形成了一张图（graph），根据距离远近、路面状况、交通路况等因素计算出对应的权值，就可以通过 Dijkstra 算法得到最优路线。此外，Dijkstra 算法在运筹学、通讯、超大型集成电路设计[5]、机器人、交通规划等领域也有应用。

2. 程序编写

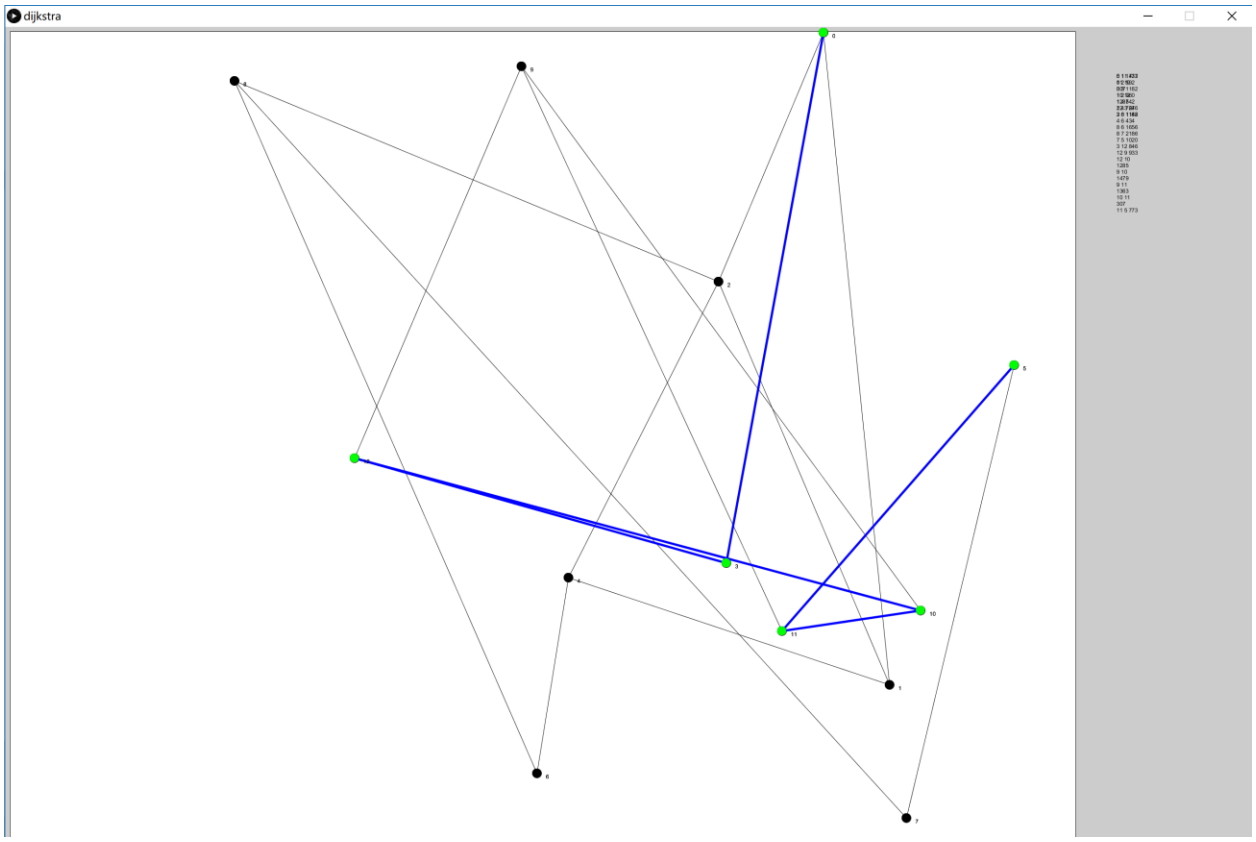
Dijkstra 程序中用到的库和 Kruskal 大体相同，唯一改动是 Node 添加了两个变量（如下）

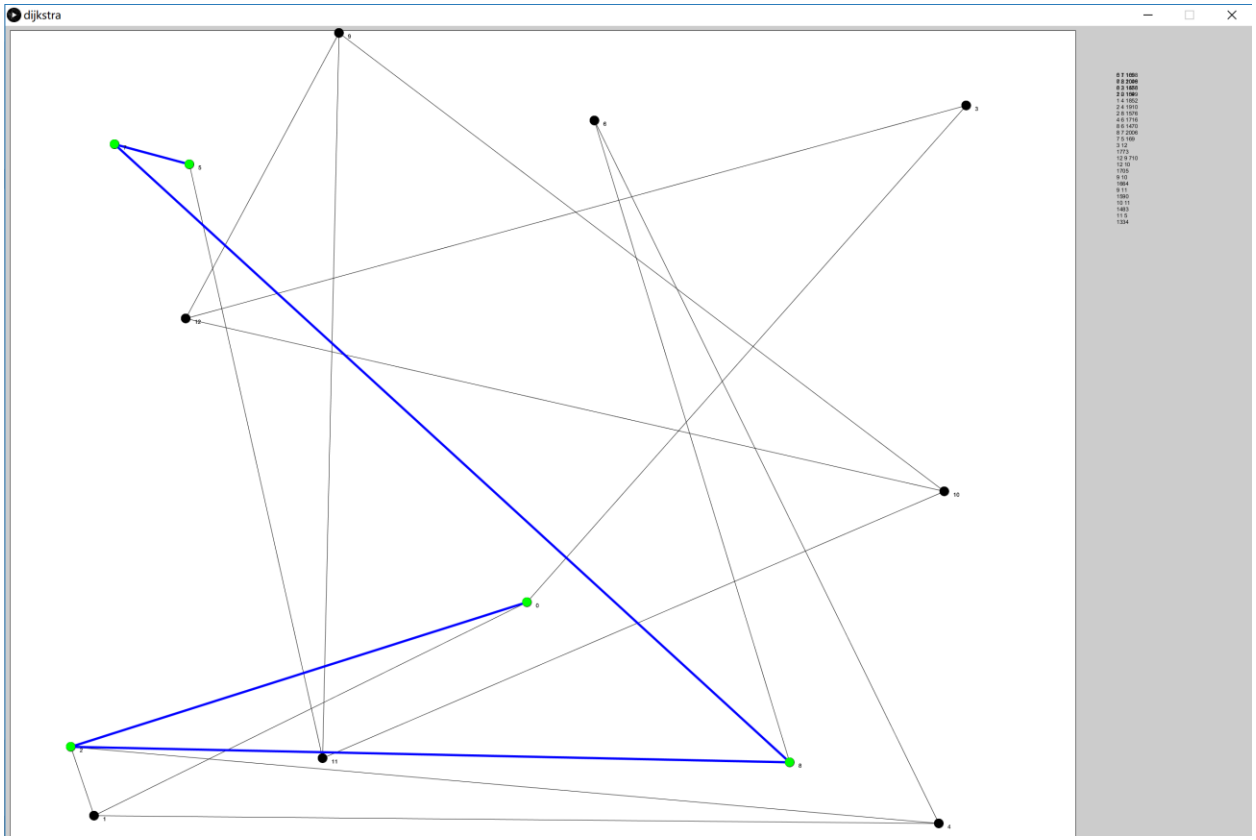
parent	Node	记端点的父节点
distance	int	端点的距离值

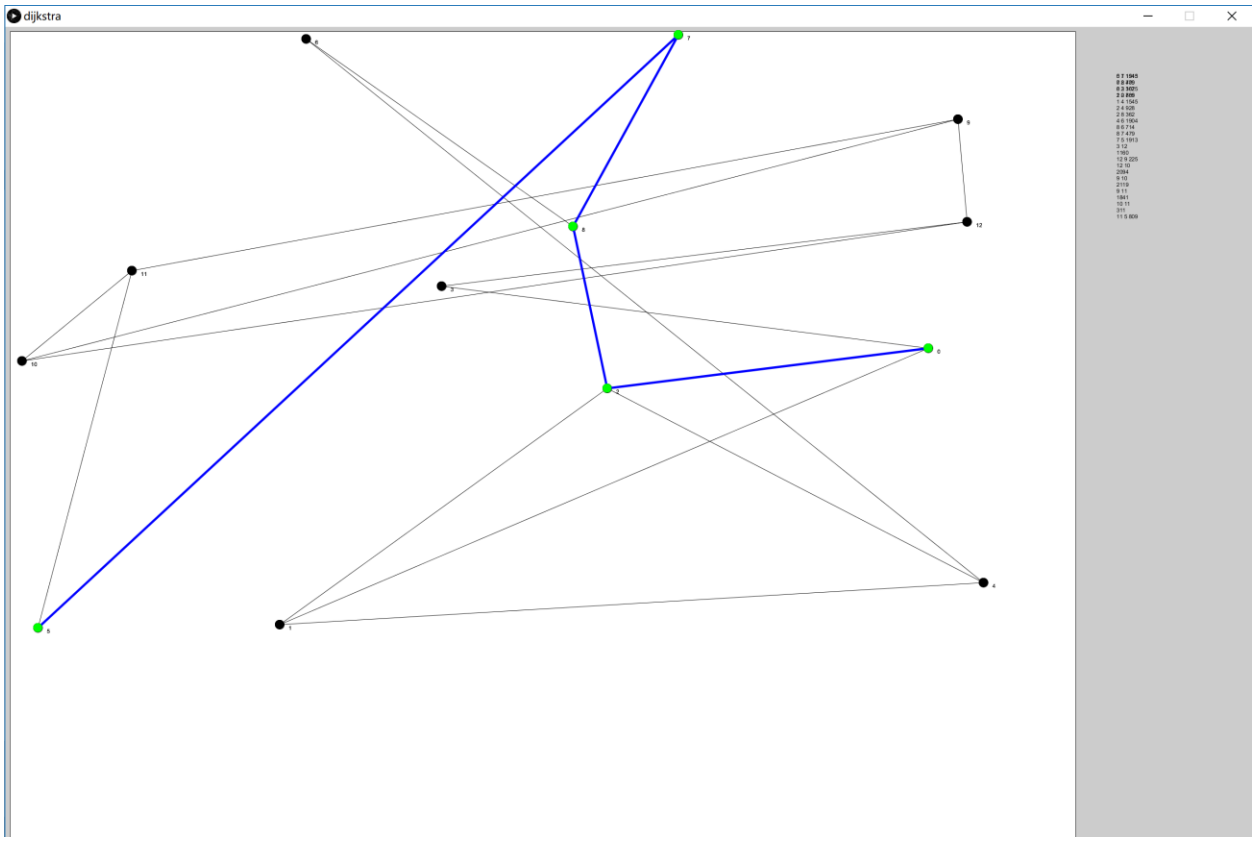
函数逻辑：

1. 定义两个图（Graph）graph 和 solution，其中 graph 用 father code 初始化。
2. 定义起点 start 和终止点 end，令 start 的距离值等于 0
3. 定义当前端点 current 等于 start
4. 当 graph 的端点数（graph.nodes.size()）不等于 0，重复运行一下部分：
   a) 将 graph 的端点按度从小到大排列
   b) 令 current 等于 graph 中的最小度点
   c) 对 current 的邻点，如果在 graph 中，
      i. 对于 current 的每个邻点 neighbor 的距离值大于该点与 current 的距离加上 current 的距离值：
         1. 令 neighbor 的距离值等于 current 的距离值加 current 与 neighbor 的距离
         2. 令 neighbor 的 parent（父节点）为 current
   d) 将 current 加到 solution 中
   e) 将 current 从 graph 中去掉
5. 打印 solution

3. 程序测试


dijkstra


dijkstra



总结与反思

本次专题研究进行的很顺利，成功的实现了 Kruskal 和 Dijkstra 算法。Processing 语言使得我能很方便地将将相关元素画出来。此外，为了实现这两个算法编写的处理图、树和二部图以及端点、边的函数库可以用于后续的研究和其他算法的实现。

在本次课题研究和后续的反思中，我有三个心得体会：首先，我体会到了面向对象编程的重大意义。如果没有面向对象编程，在处理图及其相关元素将会很麻烦，可能用到大量的数组。有了面向对象编程，只要定义出图、端点等的若干特征，就可以将图、点等作为一种数据类型直接进行操作，这大大简化了编写算法相关步骤的过程。其次，在将抽象的数学定义变为程序中的相关语句时，我体会到了贴近数学本质的重要性。例如，在判断边的两个端点时，我开始的思路是通过处理 father code 推出邻点关系，然而很快发现这个不太“美观”的方法带来很多问题，使后续的功能变得很复杂。这是，我想到了图论中边本质上就是点集的子集，如果这一本质在定义“边”的代码中能够有所体现，就会大大方便后续的工作。Father code 仅仅是图的一种编码方式，不是图的本质。有了这样的见解后，我在后续的编程中时我的代码尽量贴近相关的数学定义，这也使得我程序编写的很顺利，同时我的代码也变得很严谨。第三点体会，就是解决问题时和别人交流的重要性。在遇到无法突破的问题时，和同学的讨论是我得到新的思路，最终将问题解决。这个交流的过程也是一个相互学习的过程，参与者都能够从中有所收获。

特别鸣谢

这是笔者第一次探索通过计算机程序来构建数学模型并运行相关算法，也是笔者独立编写过的最复杂的（两个）程序。在此特别鸣谢我的同学和友人张咏境，在和他的交流中我的收获非常大，特别是在

编写 Kruskal 算法中检测两点间是否存在路径的函数时，与你的讨论使我得到新的视角，确立一条可行的方案。你的不断鼓励和对问题的热情也为我解决程序难题提供了动力。

参考文献

[1] Sneath, P. H. A. (1 August 1957). "The Application of Computers to Taxonomy". Journal of General Microbiology. 17 (1): 201–226 : http://mic.microbiologyresearch.org/content/journal/micro/10.1099/00221287-17-1-201

[2] Ohlsson, H. (2004). Implementation of low complexity FIR filters using a minimum spanning tree. 12th IEEE Mediterranean Electrotechnical Conference (MELECON 2004). 1. pp. 261–264. : http://ieeexplore.ieee.org/document/1346826/

[3] Asano, T.; Bhattacharya, B.; Keil, M.; Yao, F. (1988). Clustering algorithms based on minimum and maximum spanning trees. Fourth Annual Symposium on Computational Geometry (SCG '88). 1. pp. 252–257. : https://dl.acm.org/citation.cfm?doid=73393.73419

[4] Devillers, J.; Dore, J.C. (1 April 1989). "Heuristic potency of the minimum spanning tree (MST) method in toxicology". Ecotoxicology and Environmental Safety. 17 (2): 227–235. : http://www.sciencedirect.com/science/article/pii/0147651389900420?via%3Dihub

[5] http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf