学号 X01814031 专业 软件工程 姓名 王浩宇

实验日期 教师签字 成绩

实验报告

# 【实验名称】 最小生成树(可视化实现)

# 【实验原理】

**程序基于Python实现**

1.其中利用了**Tkinter库作为可视化界面的制作**，这是一款Tkinter 是 Python 的标准 GUI 库。Python 使用 Tkinter 可以快速的创建 GUI 应用程序。

2.利用Networkx库，它内置了常用的图与复杂网络分析算法，可以方便的进行复杂网络数据分析、仿真建模等工作。对于本次实验图的可视化这部分，Networkx+Matplotlib是一个很好的选择，操作方便而且功能多样。因此我选用了**Networkx+Matplotlib作为图的可视化呈现。**

3.python程序生成应用程序选择了**Pyinstaller ,** 它的好处是可以把python环境和库放进去一起打包，因此不会受电脑环境的问题而无法执行，缺点就是存储占用有些大。

# 【实验内容】

## 代码解析：

1. **import** networkx as nx
2. **import** matplotlib.pyplot as plt
3. **import** tkinter as tk

调用三个库：Networkx用于连接顶点形成图，matplotlib用于配合Networkx显示图象，Tkinter用于制作可视化输入台/控制台。

1. N = 10005
2. Father = [0] + [n+1 **for** n **in** range(N)]
3. R = [0] \* (N+1)
4. \_ = 1 << 10
5. graph = []
6. n = 0

定义图的相关定义。

1. **def** find(x):  # 并查集的find
2. **global** Father
3. **if** x == Father[x]:
4. **return** x
5. Father[x] = find(Father[x])
6. **return** Father[x]

定义并查集中的find，用于kruscal算法中的搜索是否已加入最小生成树，防止成环。

1. **def** union(x, y):  # 并查集的union
2. **global** Father, R
3. x = find(x)
4. y = find(y)
5. **if** R[x] > R[y]:
6. R[y], R[x] = R[x], R[y]
7. Father[x] = y
8. **if** R[x] == R[y]:
9. R[y] += 1

定义并查集中的合并，将顶点加入生成树，这里的Union还做了平衡处理，加快查找的速度。

1. **def** kruskal(G, num):
2. E = []
3. **for** u **in** range(num):
4. **for** v **in** range(num):
5. **if** u != v:
6. E.append((G[u][v], u+1, v+1))
7. T = set()
8. **for** \_, u, v **in** sorted(E):
9. **if** len(T) == num-1:
10. **break**
11. **if** find(u) != find(v):
12. T.add((u, v))
13. union(u, v)
14. **return** T

kruscal算法，用作寻找最小生成树，返回一个集合Set，该集合即为最小生成树的边集。

1. window=tk.Tk()
2. window.title('Minimum Spanning Tree(kruscal)')
3. window.geometry('700x700')

创建一个窗口，窗口名为：Minimum Spanning Tree(kruscal)，窗口大小为：700x700

1. I=tk.Label(window,
2. text="please input the number of nodes:",
3. font=('Arial',12),
4. width=40,height=2)
5. I.pack()

创建提示字模块，在窗口上显示：please input the number of nodes:

字体大小为12，字体类型为Arial，字模块宽40，高2

1. numofnodes = tk.Entry(window)
2. numofnodes.pack()

创建一个文本输入框，放在提示字下方

1. L2=tk.Label(window,
2. text="please input the Adjacency Matrix",
3. font=('Arial',12),
4. width=40,height=2)
5. L2.pack()

创建提示字模块，在窗口上显示：please input the Adjacency Matrix

字体大小为12，字体类型为Arial，字模块宽40，高2

1. data\_mat = tk.Text(window,height=10)
2. data\_mat.pack()

创建一个文本输入框，放在提示字下方

1. **def** insert\_data():
2. **global** n
3. n = int(numofnodes.get())
4. **print**(n)
5. tep = data\_mat.get("0.0", "end").strip()
6. graph=[[int(i) **for** i **in** row.split(' ')]**for** row **in** tep.split('\n')]
7. #graph.pop()  # 列表最后一个元素是空删除它
8. Gbefore = nx.Graph()
9. **print**(graph[0][0])
10. **for** i **in** range(n):
11. **for** j **in** range(n):
12. **if** (graph[i][j] != 0):
13. Gbefore.add\_edges\_from([(i + 1, j + 1)])
14. **else**:
15. graph[i][j] = \_
16. plt.figure(figsize=(15, 7))
17. plt.subplot(1, 2, 1)
18. plt.title("The input graph :")
19. nx.draw(Gbefore, with\_labels=True, edge\_color='black', node\_color='orange', node\_size=1000)
20. G = nx.Graph()
21. G.clear()
22. # 导入所有边，每条边分别用tuple表示
23. # a\_list = []
24. G.add\_edges\_from(list(kruskal(graph, len(graph))))
25. **print**(len(graph))
26. plt.subplot(1, 2, 2)
27. plt.title("Minimum Spanning Tree's picture after using Kruscal")
28. nx.draw(G, with\_labels=True, edge\_color='black', node\_color='orange', node\_size=1000)
29. plt.show()

用作button的command功能：

1.输入数据，对接收到的文本框字符进行切片，转换为二维列表，同时要注意忽略结尾的空格或者回车，利用stricp()函数即可。

2.对原图进行绘制：利用networkx的add\_edges\_from功能，并用Matplotlib来进行图像的显示。

3.执行kruscal算法，计算最小生成树。

4.对最小生成树进行绘制：利用networkx的add\_edges\_from功能，并用Matplotlib来进行图像的显示。

1. b=tk.Button(window,text='Run',width=15,height=4,command=insert\_data)
2. b.pack()
4. window.mainloop()

创建一个按钮：

显示名字为Run

按钮大小为：长15，高4。

点击后触发 Insert\_data函数的功能

Window.mainloop()是让窗口不关闭

# 【小结或讨论】

在我一选到这门课的时候，实验室的学长就和我说杜老师的课有大作业，当时感到很有意思。

到了实验题目出来时，我列出了两个个方向去实现可视化界面：c++QT和python+库。因为对c++比较熟悉，初步选择的是c++QT，但是被语法劝退了，先学QT的话最少得一个星期多才能入门，我就去选择了python的方案。

同样也是零基础，只好先利用的python写出了kruscal的算法。然后花了一天来学习Tkinter来实现可视化控制台，摸索着查阅官网的文档和一些教学视频，终于学会一部分我可以用到的界面实现方法。然后又用了半天的时间学习Networkx库，说起来也是很幸运，这个假期看一门深度学习科普视频时有看到这个库用来显示神经网络层，我抱着试试的心态查询了下这个库，果真也有绘图的功能，于是半天看了些官方的文档，学到了一些基本要用到的函数。最后花了半天来用所学到的知识来做可视化界面。

我一直想做一个可视化界面的小软件，一直没机会去实现，很荣幸在这节课上完成了愿望，两天的时间学习了很多的东西，受益匪浅，我相信这些知识会在我未来研究生期间非常有用，也感谢老师给我们一个学习新知识的机会。由于时间并不充裕，这学期由于疫情课程安排十分紧凑，这个小软件做的还有很多缺陷，如果暑假有时间，我会继续完善它的其他功能，比如显示其他数据结构的可视化，希望最后这个软件可以用作教学示范，到时候我会repo到github的仓库，记录一下这一段自己的成长。

最后，再次感谢老师的让我们做小项目的用意以及用心的选题，以后有机会的话我会推荐给实验室学弟这门有趣的课。

# 【源代码】

1. **import** networkx as nx
2. **import** matplotlib.pyplot as plt
3. **import** tkinter as tk
5. # 调用三个库：Networkx用于连接顶点形成图
6. # matplotlib用于配合Networkx显示图象
7. # Tkinter用于制作可视化输入台/控制台。
9. N = 10005
10. Father = [0] + [n+1 **for** n **in** range(N)]
11. R = [0] \* (N+1)
12. \_ = 1 << 10
13. graph = []
14. n = 0
15. #定义图的相关定义。
16. #text\_content = []#用来储存每一行内容的列表


20. # 定义并查集中的find
21. # 用于kruscal算法中的搜索是否已加入最小生成树，防止成环。
23. **def** find(x):  # 并查集的find
24. **global** Father
25. **if** x == Father[x]:
26. **return** x
27. Father[x] = find(Father[x])
28. **return** Father[x]


32. # 定义并查集中的合并，将顶点加入生成树
33. # 这里的Union还做了平衡处理，加快查找的速度。
34. **def** union(x, y):  # 并查集的union
35. **global** Father, R
36. x = find(x)
37. y = find(y)
38. **if** R[x] > R[y]:
39. R[y], R[x] = R[x], R[y]
40. Father[x] = y
41. **if** R[x] == R[y]:
42. R[y] += 1

45. # kruscal算法，用作寻找最小生成树，返回一个集合Set，该集合即为最小生成树的边集。
46. **def** kruskal(G, num):
47. E = []
48. **for** u **in** range(num):
49. **for** v **in** range(num):
50. **if** u != v:
51. E.append((G[u][v], u+1, v+1))
52. T = set()
53. T.clear()
54. **for** \_, u, v **in** sorted(E):
55. **if** len(T) == num-1:
56. **break**
57. **if** find(u) != find(v):
58. T.add((u, v))
59. union(u, v)
60. **return** T

63. # 用作button的command功能：
64. # 1.输入数据，对接收到的文本框字符进行切片，转换为二维列表，同时要注意忽略结尾的空格或者回车，利用stricp()函数即可。
65. # 2.对原图进行绘制：利用networkx的add\_edges\_from功能，并用Matplotlib来进行图像的显示。
66. # 3.执行kruscal算法，计算最小生成树。
67. # 4.对最小生成树进行绘制：利用networkx的add\_edges\_from功能，并用Matplotlib来进行图像的显示。

70. **def** insert\_data():
71. **global** n
72. n = int(numofnodes.get())
73. **print**(n)
74. tep = data\_mat.get("0.0", "end").strip()
75. graph=[[int(i) **for** i **in** row.split(' ')]**for** row **in** tep.split('\n')]
76. #graph.pop()  # 列表最后一个元素是空删除它
77. Gbefore = nx.Graph()
78. **print**(graph[0][0])
79. **for** i **in** range(n):
80. **for** j **in** range(n):
81. **if** (graph[i][j] != 0):
82. Gbefore.add\_edges\_from([(i + 1, j + 1)])
83. **else**:
84. graph[i][j] = \_
85. plt.figure(figsize=(15, 7))
86. plt.subplot(1, 2, 1)
87. plt.title("The input graph :")
88. nx.draw(Gbefore, with\_labels=True, edge\_color='black', node\_color='orange', node\_size=1000)
89. G = nx.Graph()
90. G.clear()
91. # 导入所有边，每条边分别用tuple表示
92. # a\_list = []
93. G.add\_edges\_from(list(kruskal(graph, len(graph))))
94. **print**(len(graph))
95. plt.subplot(1, 2, 2)
96. plt.title("Minimum Spanning Tree's picture after using Kruscal")
97. nx.draw(G, with\_labels=True, edge\_color='black', node\_color='orange', node\_size=1000)
98. plt.show()
100. # 创建一个窗口，
101. # 窗口名为：Minimum Spanning Tree(kruscal)，
102. # 窗口大小为：700x700
103. window=tk.Tk()
104. window.title('Minimum Spanning Tree(kruscal)')
105. window.geometry('700x700')
107. # 创建提示字模块，
108. # 在窗口上显示：please input the number of nodes:
109. # 字体大小为12，字体类型为Arial
110. # 字模块宽40，高2
112. I=tk.Label(window,
113. text="please input the number of nodes:",
114. font=('Arial',12),
115. width=40,height=2)
116. I.pack()

119. # 创建一个文本输入框，放在提示字下方
121. numofnodes = tk.Entry(window)
122. numofnodes.pack()
124. # 创建提示字模块，
125. # 在窗口上显示：please input the Adjacency Matrix
126. # 字体大小为12，字体类型为Arial，
127. # 字模块宽40，高2
129. L2=tk.Label(window,
130. text="please input the Adjacency Matrix",
131. font=('Arial',12),
132. width=40,height=2)
133. L2.pack()
135. #创建一个文本输入框，放在提示字下方
136. data\_mat = tk.Text(window,height=10)
137. data\_mat.pack()

140. # 创建一个按钮：
141. # 显示名字为Run
142. # 按钮大小为：长15，高4。
143. # 点击后触发 Insert\_data函数的功能
145. b=tk.Button(window,text='Run',width=15,height=4,command=insert\_data)
146. b.pack()
148. # Window.mainloop()是让窗口不关闭
150. window.mainloop()