计算机科学与技术学院可视化技术实验报告

实验题目：用 RT 算法实现 radial tree layout		学号：201900150221
日期：10.18	班级： 19 智能	姓名： 张进华
Email：zjh15117117428@163.com		

实验目的：

用 RT 算法实现 radial tree layout

实验软件和硬件环境：

Visual studio Code python 3.9.7

Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz

实验原理：

实验步骤

步骤一 构造数据集

构造数据集如下，实现方式为字典，根节点下包含子树，键为父节点名，包含字典列表包含其子节点，实现代码如下：

```
# 构造数据集
data = [
{
 "name": "root",
 "children": [
  {
   "name": "animate",
   "children": [
    {"name": "Easing", "value": 17010},
    {"name": "FunctionSequence", "value": 5842},
    {
     "name": "interpolate",
     "children": [
      {"name": "ObjectInterpolator", "value": 1629},
      {"name": "PointInterpolator", "value": 1675},
      {"name": "RectangleInterpolator", "value": 2042}
     ]
    },
```

步骤二 定义树节点类

定义类 Node，包含树节点的各种信息以及搜寻树节点的父节点、左兄弟节点以及添加子节点的函数

```python
# 3.定义树节点
treeNodes={}
class Node:
    def __init__(self,par_name=None,y=0,left_bro=None,name=None):
        self.width=0 # 树节点大小
        self.parent=par_name
        self.x=None
        self.y=y
        self.children=[]
        self.left_bro=left_bro # 树结点的左右兄弟
        # 树节点的最左右节点坐标值
        self.x_left={1:500,2:500,3:500,4:500}
        self.x_right={1:-500,2:-500,3:-500,4:-500}
        self.offset=0 # 合并时移动步长
        self.name=name
    # 添加子节点
    def add_child(self,ch):
        self.children.append(ch)
    # 寻找左兄弟
    def get_lbrother(self):
        if self.left_bro == None:
            return None
        return treeNodes[self.left_bro]
    # 寻找父节点
    def get_parent(self):
        return treeNodes[self.parent]
```

0.5s

步骤三 构造树，定义添加树节点函数

```python
# 4.定义添加树节点函数
def appendNode(data,par_name=None,y=0,left_bro=None):
    name=data['name']
    # 将当前节点加入树节点
    if name not in treeNodes.keys():
        treeNodes[name]=Node(par_name,y,left_bro,name)
    else:
        return
    if 'children' in data.keys():
        children=data['children']
    else:
        return

    if len(children)!=0:
        left=None
        for t in children:
            # 遍历子节点
            treeNodes[name].addchild(t['name'])
            appendNode(t,name,treeNodes[name].y+1,left_bro=left)
            left=t['name']

appendNode(data)
```

步骤四 初始化

初始化，遍历节点，获取每个节点即当前层节点需要合并时的移动距离，其父节点坐标为左右子节点的坐标均值，而移动距离则是遍历所有节点，找出保证不造成重复情况下的最大距离，使其紧凑

```
# 5.初始化
def initial(node):
    # 获取当前节点属性
    node.x=0
    y=node.y
    if len(node.children)==0:
        if node.y!=0:
            node.width=1/node.y # 节点宽度
            node.x_left[y]=0 # 初始化左节点坐标
            node.x_right[y]=node.width # 初始化右节点坐标
            node.x=node.width/2 # x坐标
    else:
        if node.y!=0:
            node.width=1/node.y
        # 遍历子节点，获取当前层节点合并需移动距离
        for k in range(len(node.children)):
            children=treeNodes[node.children[k]]
            initial(children)
            # 叶子节点
            if k==0:
                for i in range(y+1,5):
                    node.x_left[i]=children.x_left[i]
            for i in range(y+1,5):
                node.x_left[i]=min(node.x_left[i],children.x_left[i])
                node.x_right[i]=max(node.x_right[i],children.x_right[i])
        # 获取最左右节点坐标
        node.x_left[y]=(treeNodes[node.children[0]].x+treeNodes[node.children[-1]].x+treeNodes[node.children[-1]].offset)/2
        node.x_right[y]=node.x_left[y]+node.width
        node.x=node.x_left[y]
    left_bro=node.get_lbrother() # 获取左兄弟节点
    offset=0
    if left_bro!=None:
        offset=0
        for i in range(y,5):
            offset=max(offset,left_bro.x_right[i]-node.x_left[i]) # 更新移动距离

    for i in range(y,5):
        node.x_left[i]+=offset
        node.x_right[i]+=offset
    node.offset=offset
```

步骤五 节点左移

遍历所有节点，根据上一步计算出的移动参数递归将所有节点的坐标更新

```
# 6.移动节点
def second_step(node,disp):
    node.x+=node.offset+disp
    for i in node.children:
        # 递归子节点
        second_step(treeNodes[i], disp+node.offset)
    node.offset=0
```

步骤六 寻找映射距离

遍历寻找树的直径，便于后续将 x 坐标映射成角度时的映射

```
# 7.获取树的直径
max_x=0
for i in range(1,5):
    max_x=max(max_x,treeNodes['root'].x_right[i])
    print(max_x)
```

步骤六 定义绘制树节点函数

将节点 x 坐标映射到角度进行绘制

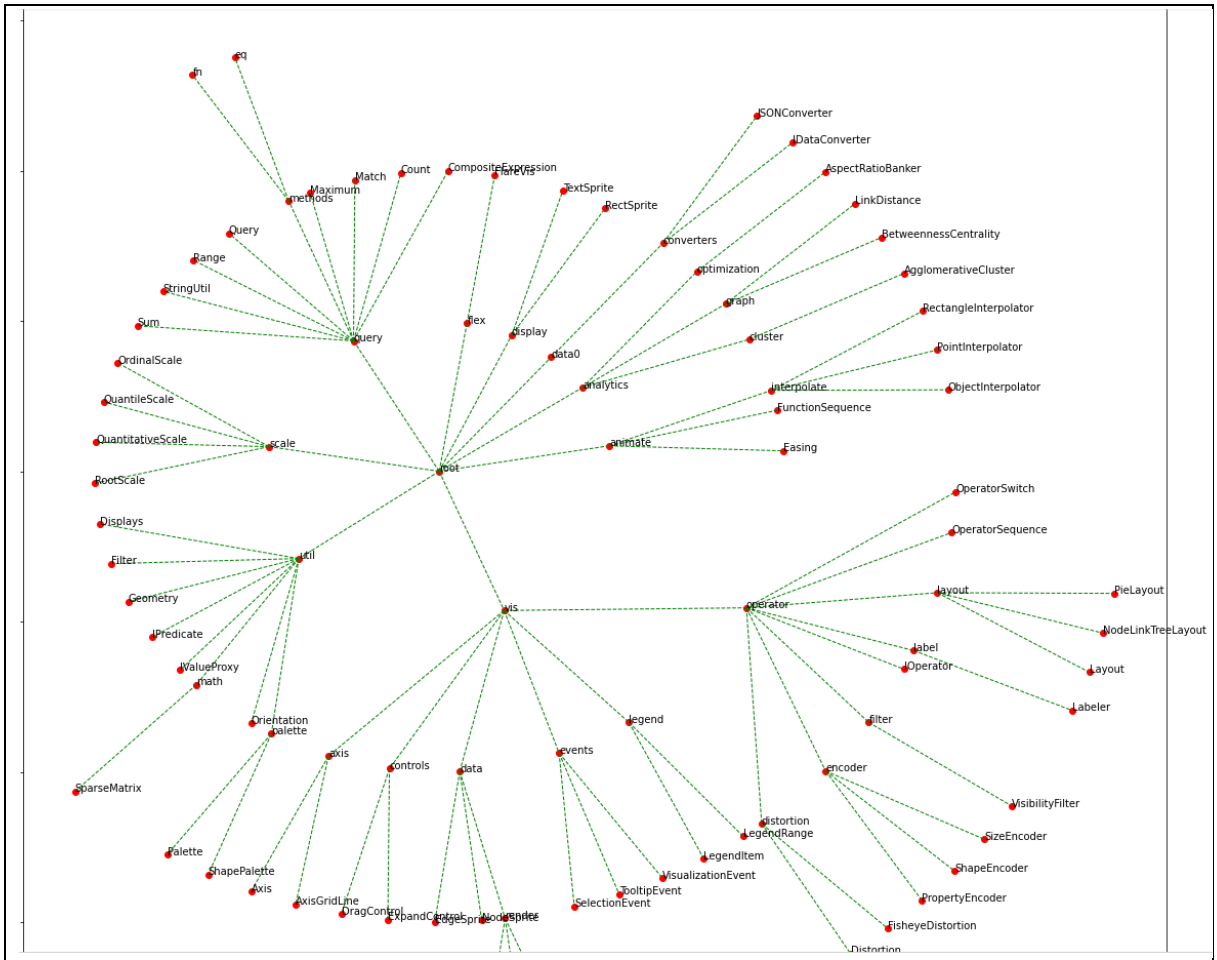
```
# 8.绘制树节点
def draw_node(node,maxx,r):
    x,y=node.x,node.y
    angle = 2 * np.pi * x / (maxx) # 角度映射
    y, x = y * np.sin(angle)*r, y * np.cos(angle)*r
    # rotation, alignment = get_label_rotation(angle, np.pi/2)

    plt.scatter(x, y, facecolor='red',marker="o")
    plt.text(x, y, node.name, fontsize=10,)
    for child in node.children:
        draw_node(treeNodes[child], maxx,r)
```

步骤七 定义会直接点连线函数

```
# 9.绘制连线
def draw_line(node, maxx,r):
    x,y=node.x,node.y
    angle = 2 * np.pi * node.x / (maxx)
    y, x = y * np.sin(angle)*r, y * np.cos(angle)*r
    for i in node.children:
        child=treeNodes[i]
        childx = child.x
        childy = child.y
        angle = 2 * np.pi * childx / (maxx)
        childy, childx = childy * np.sin(angle)*r, childy * np.cos(angle)*r
        plt.plot([x, childx], [y, childy]
        , linestyle='dashed', linewidth=1, color="green")
        draw_line(child, maxx,r)
```

步骤八 实现效果

最终效果如下



结论分析与体会：

RT 算法主要要满足以下几点：

1. 节点不交叉
2. 同层节点要在同一水平线
3. 节点尽可能靠近
4. 父亲在两个孩子中间（二叉树）

几种加速方法：

1. 给每个节点类定义变量 mod, 储存其向右移动的距离
2. 给每个节点类定义变量 thread，加速 contour 函数计算左右子树间距