实验题目

- 用RT算法实现radial tree layout

实验日期

- 2021.10.25

班级

- 19智能

姓名

- 倪诗宇

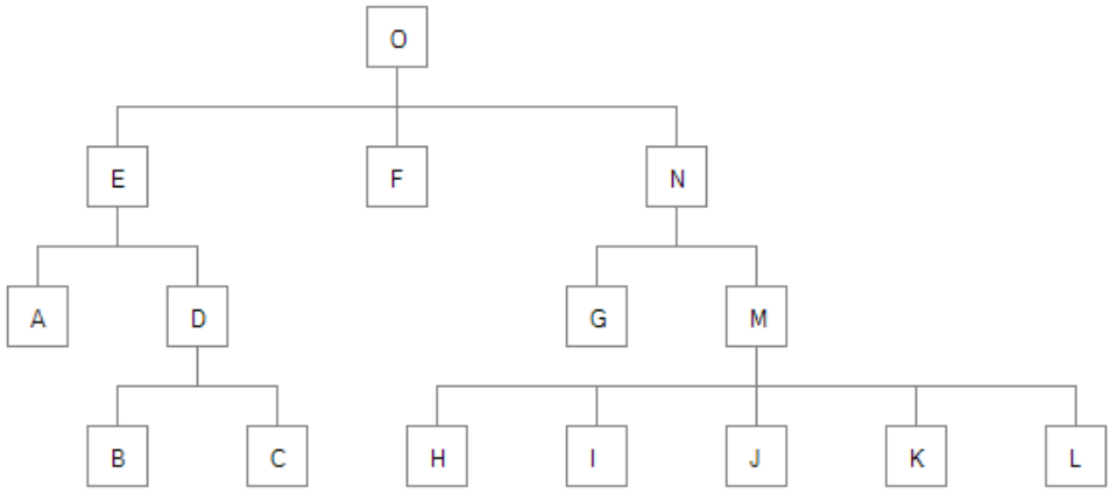
实验目的

- 用RT算法实现radial tree layout

实验过程

RT的原理

例子



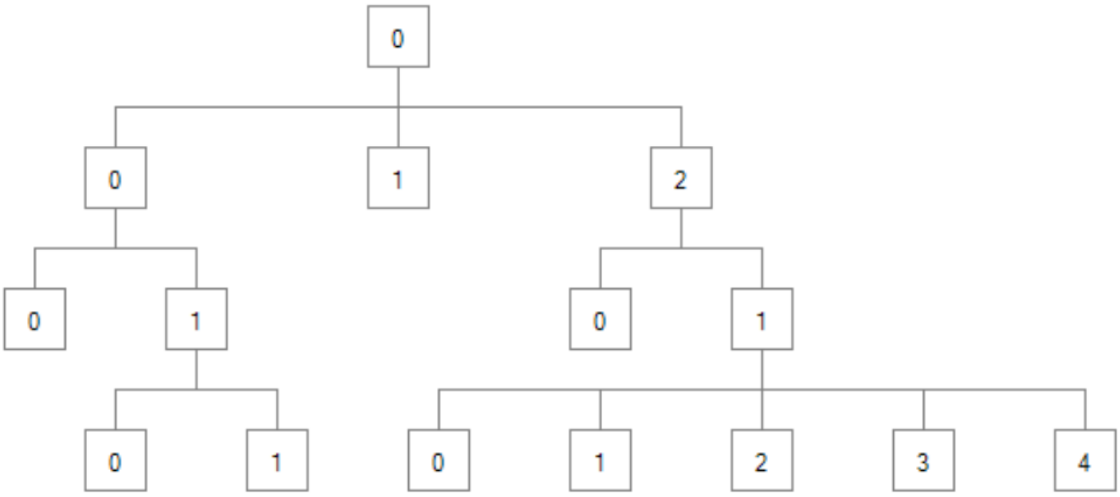
- y的值就是节点所在的树的深度
- x值的获取
    - 对树进行后序遍历
        - 如果一个节点在最左边，则令x = 0。否则，令x = leftSibling.x + 1
        - 我们想要每个父亲都在其孩子的上方居中，是第一个孩子的 X 位置和最后一个孩子的 X 位置之间的中间点。
            - 如果父节点没有左兄弟节点，则将其 X 值更改为此中点值

        - 如果它有一个左兄弟，我们将把它存储在另一个节点属性中(mod)
            - Mod 属性用于确定修改子节点的 X 值以将它们置于父节点下方的程度，应该设置为 Parent.X – MiddleOfChildrenX 以确定移动孩子的正确距离
        - 检查此树是否与之前的任何兄弟树冲突。这意味着循环遍历当前子树中的每个 Y 级别，并检查节点左侧任何兄弟节点为根的子树的最右侧 X 值是否与当前子树中任何子节点的最左侧 X 值交叉。有交叉则移动当前节点，并根据需要调整 Mod 属性，以确定子节点的移动量
    - 再次遍历树以确定不会在屏幕外绘制任何子项，并根据需要调整 Mod 属性。如果 Mod 属性为负，就会发生这种情况
    - 对树进行第三次遍历以确定每个节点的最终 X 值。这将是节点的 X，加上该节点的所有父节点的所有 Mod 值的总和

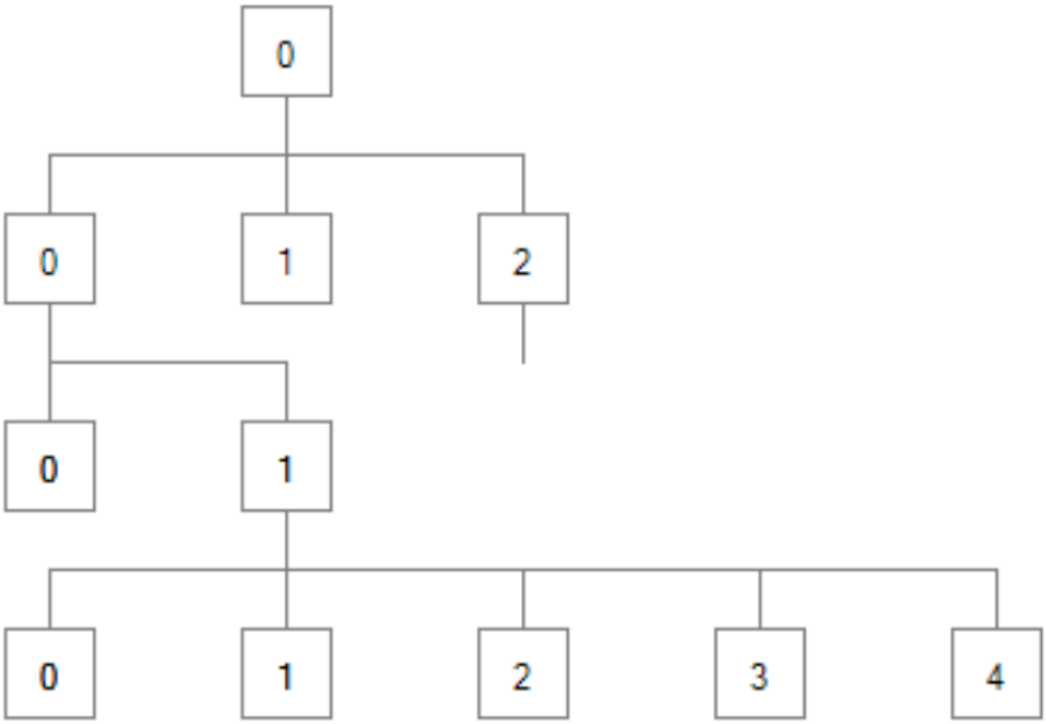
第一次遍历

- 为 的每个节点分配一个 X 值，如果有左兄弟，为 leftSibling.X + 1，否则为0
- 将子节点移到父节点下
- 验证没有孩子与其他树冲突

初始化



如果按实际绘制，则



可以看到，有些点有重叠，且看不出树的层次结构。下一步将子节点定位在其父节点下

在父节点下定位子节点

首先，找到将父节点置于其子节点上的 X (desired_X)值。

- 如果有 1 个孩子，则所需的 X 值与孩子的 X 值相同。
- 如果有多个孩子，取第一个孩子的X和最后一个孩子的X，并找到两者之间的中点。

接下来，检查这个父节点是否在它的左边有任何兄弟节点。

- 如果没有，则设置当前节点的 X = desired_X(移动父节点)
- 否则，令该节点的Mod = X - desired_X , 以移动子节点，使父节点位于其上方中央

但是这不能避免节点重叠问题，因此下面我们检查节点之间是否有冲突

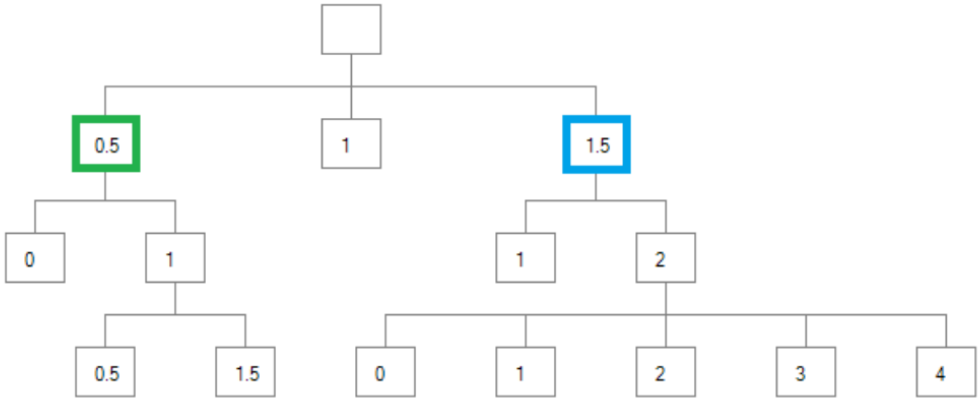
检查节点冲突

如果一个节点有任何子节点，我们需要遍历所有级别的子节点，并确保分配给该子节点的 X 值不与分配给同一级别先前定位的子节点的任何 X 值冲突

为此，我们记录Contour（边框）

- 我们遍历当前节点的所有子节点，并为每个 Y 值记录最小 X 位置（记录当前子树所有深度的左边界）
- 对于这个节点左边的每个兄弟节点，我们循环遍历它的所有子节点并记录每个 Y 处的最大 X 位置（记录左边的所有兄弟子树的右边界）

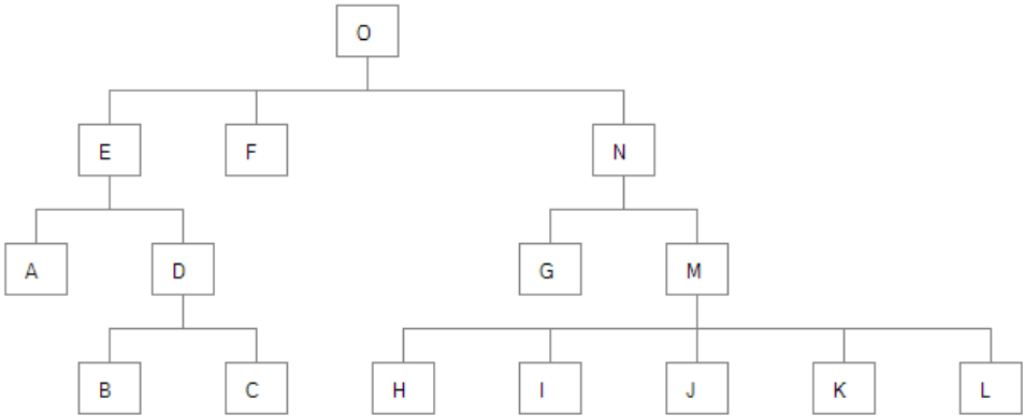
因此，在处理节点 N（蓝色）时，其左轮廓为`{ 1.5, 1, 0.0 }`，而节点 E（绿色）的右轮廓为`{ 0.5, 1.0, 1.5 }`。



我们可以看到第一层和第二层重叠

因此，将蓝色点及其子树向右移，以至于不再重叠

- 这样也会带来问题，就是可能间距太大，中间有很多空白

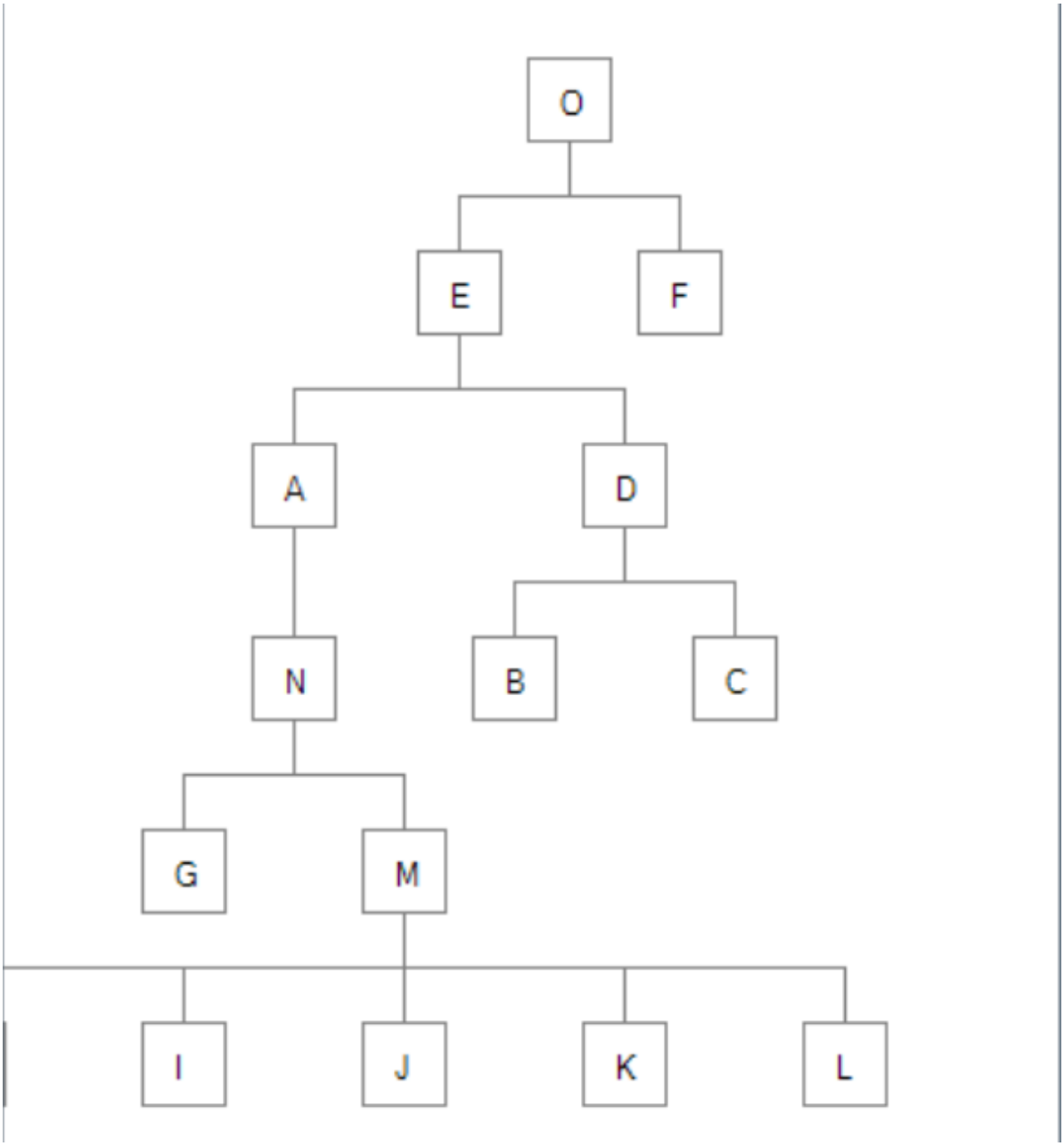


- 注意这里的 F 就是
- 为此，我们将移动的距离除以两个冲突节点之间的节点数 + 1，并将每个中间节点移动此值。
  - 在我们的例子中，我们将节点移动了 2.5，并且节点 E 和 N 之间只有 1 个节点 (F)，因此我们需要将 F 移动 2.5 / (1 + 1) = 1.25

第二次遍历

一些节点可以有一个负的 Mod 值，这也可以将子节点的最终 X 定位为负值，并将它们渲染到屏幕外

将上面的例子中的节点 N 移动到 A 下，H就会画在外面



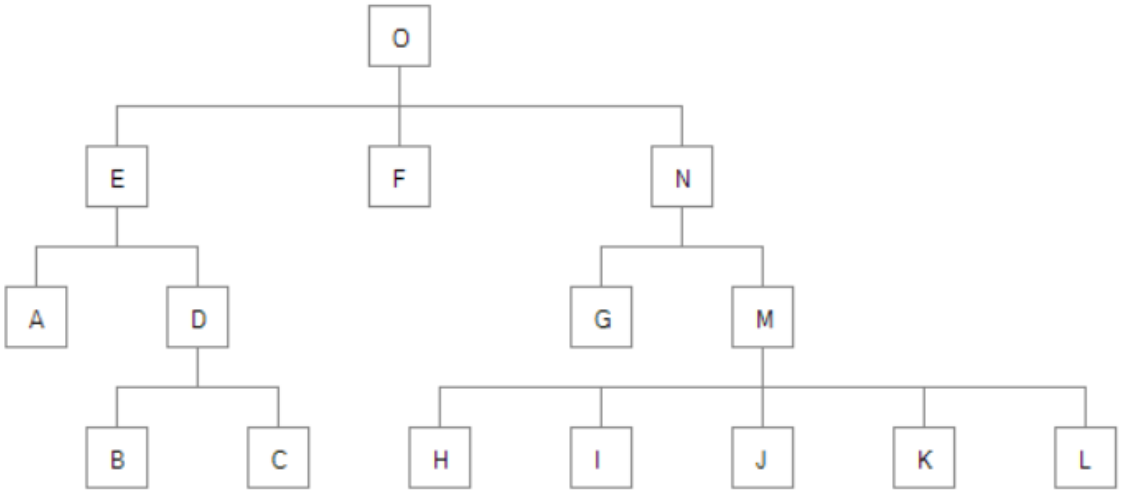
解决这个问题的最好方法是将根节点移动足够多，这样最终的 X 值都不会为负

为此，我们可以获取根节点的左轮廓，找到最小的 X 值，如果为负值，我们将根节点移动该量，同时改变Mod

最后一次遍历

第三次树遍历通过将其所有父节点的 Mod 值与其 X 值相加来计算每个节点的最终 X 位置。（上层的移动会传递到下层）

最终结果



在进行实验的过程中，首先通过以上三次遍历，得到一颗正常的树

基本框架代码如下：

```
def firstwalk(v, distance=1.):
    if len(v.children) == 0:
        if v.lmost_sibling:
            v.x = v.lbrother().x + distance
        else:
            v.x = 0.
    else:
        default_ancestor = v.children[0]
        for w in v.children:
            firstwalk(w)
            default_ancestor = apportion(w, default_ancestor, distance)
        #print "finished v =", v.tree, "children"
        execute_shifts(v)

        midpoint = (v.children[0].x + v.children[-1].x) / 2

        ell = v.children[0]
        arr = v.children[-1]
        w = v.lbrother()
        if w:
            v.x = w.x + distance
            v.mod = v.x - midpoint
        else:
            v.x = midpoint
    return v
```

```
def second_walk(v, m=0, depth=0, min=None):
    v.x += m
    v.y = depth
    if min is None or v.x < min:
        min = v.x
    for w in v.children:
        min = second_walk(w, m + v.mod, depth+1, min)
    return min
```

```
def third_walk(tree, n):
    tree.x += n
    for c in tree.children:
        third_walk(c, n)
```

然后将其坐标转换为弧度制，以实现 radial tree layout

```
def width(apex,xm=0):#算max_x ， 也就是整棵树的宽度
    if not apex.children:
        return xm
    for child in apex.children:
        if child.x > xm:
            xm = child.x
            #print xm
        xm = width(child,xm)
    return xm
```

```
def angleCo(x,y,xm):#求点的弧度坐标
    angle=2*np.pi*x/(xm + 1)
    nx,ny=y*np.sin(angle), y*np.cos(angle)
    return nx,ny
```

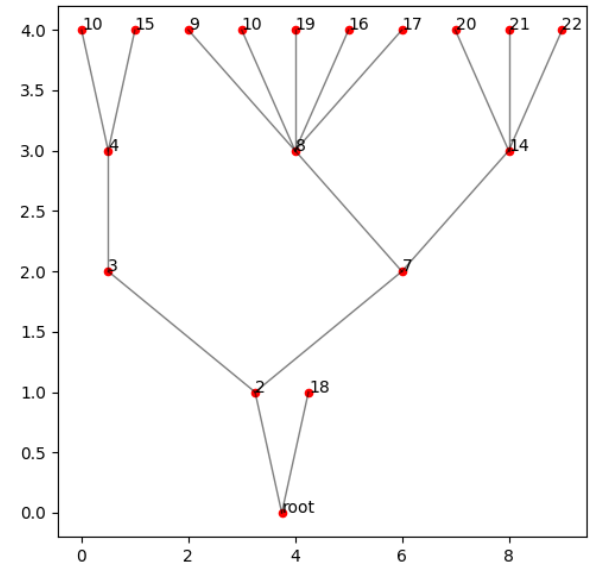
```
def drawt(root,circle):#画点
    x=root.x
    y=root.y
    if circle == True:
        x,y=angleCo(x,y,max_x)
    plt.scatter(x, y, facecolor='red',lw = 0,s=30)
    plt.text(x, y,root.tree,fontsize=10)
    for child in root.children:
        drawt(child,circle)
```

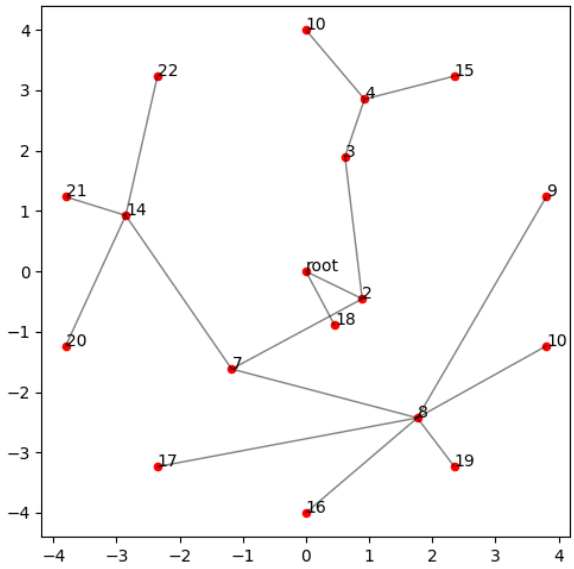
```
def drawconn(root,circle):#画连接线
    rootx=root.x
    rooty=root.y
    if circle == True:
        rootx,rooty=angleCo(rootx,rooty,max_x)
    for child in root.children:
        childx=child.x
        childy=child.y
        if circle == True:
            childx,childy=angleCo(childx,childy,max_x)
        plt.plot([rootx, childx],[rooty,childy],linestyle='-
',linewidth=1,color='black',alpha=0.5)
        drawconn(child,circle)
```

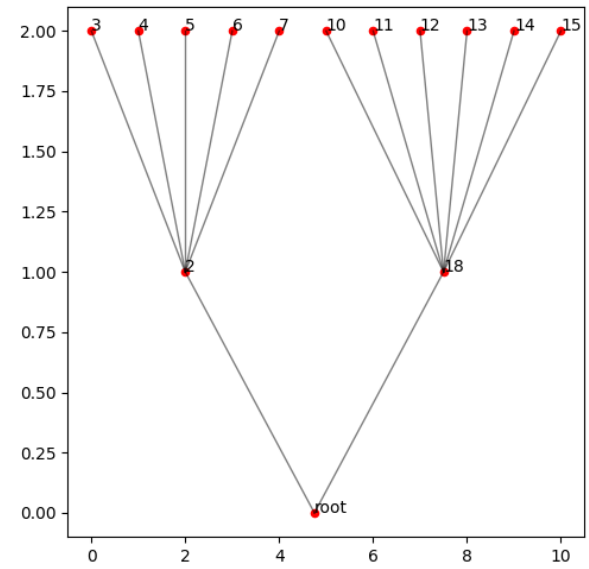
最后使用两个样例画出结果

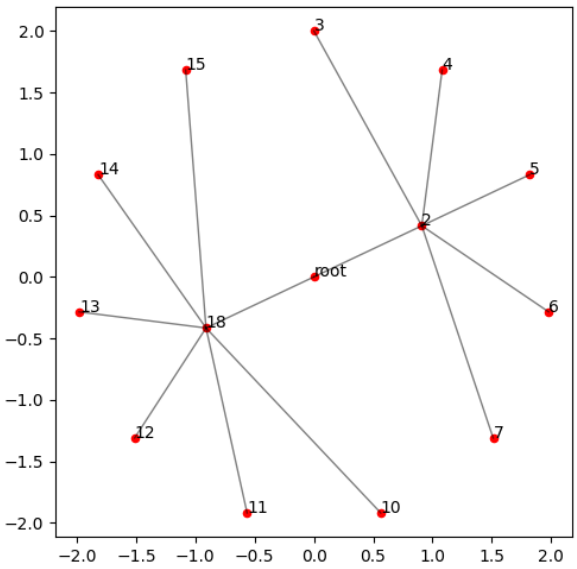
样例通过下面的代码输入

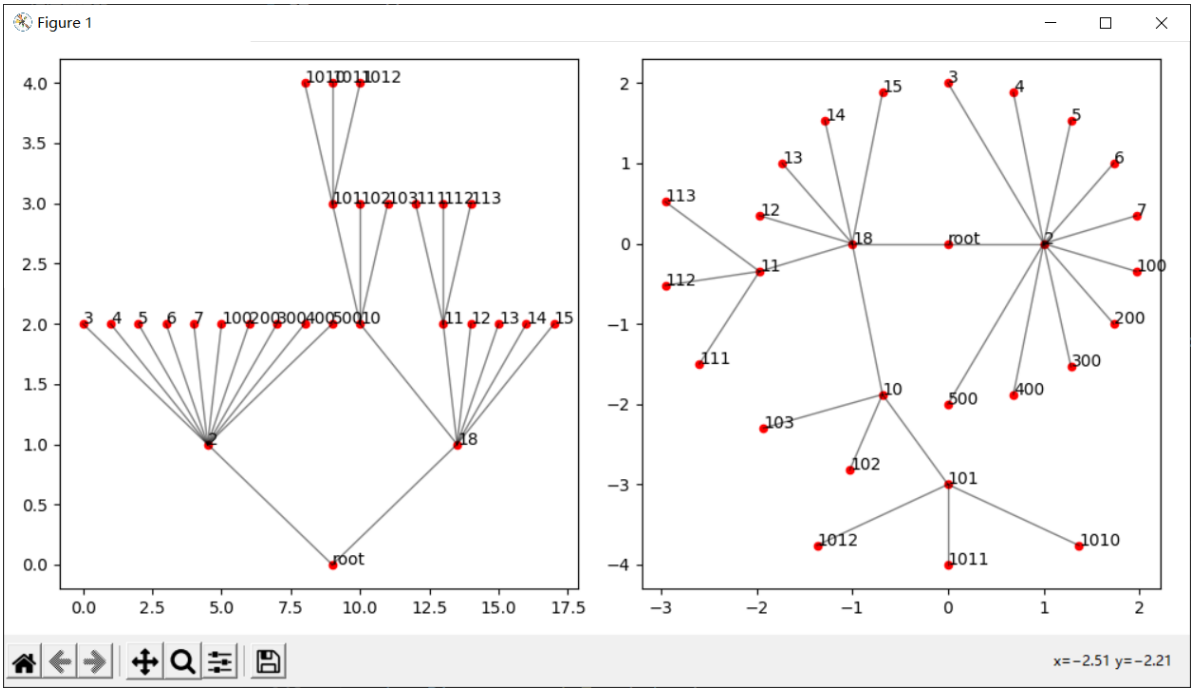
```
edges={'root':['2','18'],
         '2':['3','4','5','6','7'],
         '18':['10' , '11','12','13','14','15']
         }#这里是第二个图的样例
```











结论分析与体会

主要学习了RT算法的原理

第一次遍历

后序遍历

- 初始化所有节点
  - 如果是最左边的点，则 X = 0
  - 否则 X = leftSibling.X + 1
- 将父亲放在孩子上方中间(Mod用来记录孩子的移动，以防每次都移动孩子提高算法复杂度。最后一起移动)
  - 如果只有一个孩子，令Parent.X = Child.X
  - 如果有多个孩子，算出孩子的desired.x = (minChild.X + maxChild.X) / 2
    - 如果父亲没有左兄弟，令Parent.X = desired.X（移动父亲）
    - 否则Parent.Mod = (Parent.x - desired.x)（记下来，用于移动孩子）
- 查看是否有重叠区域（利用Contour）
  - 当前节点的子树的所有深度的最左端
  - 右边节点的子树的所有深度的最右端
  - 如果有重叠部分，将当前节点为根的子树整体右移。（只在当前节点的 x 上加， 孩子的移动加在当前节点的Mod上）
    - 这样可能会造成中间有空白
      - 将两个节点间的兄弟节点均匀右移（有孩子的要修改Mod）

第二次遍历

- 判断最左端是否小于0
- 小于0则移动，子节点的移动同样放在mod上

最后一次遍历

- 所有孩子节点的 X 都加上其父节点的 Mod