山东大学 计算机科学与技术 学院

可视化技术 课程实验报告

学号：201900130176	姓名： 李伟国	班级： 智能
实验题目： RT 算法实现 radial tree layout		
实验学时：	实验日期： 2021/10/23	

实验目的：

硬件环境：

处理器：AMD Ryzen 5 3600 6-Core Processor 3.60 GHz

Ram 16.0 GB

软件环境：

实验步骤与内容：

Tree 的表示方法：

1）Identation（windows 文件夹）

2）节点表示法（根节点，子节点，叶子节点）

3）闭包：表示层次结构

4）分层法（相对位置和对齐）

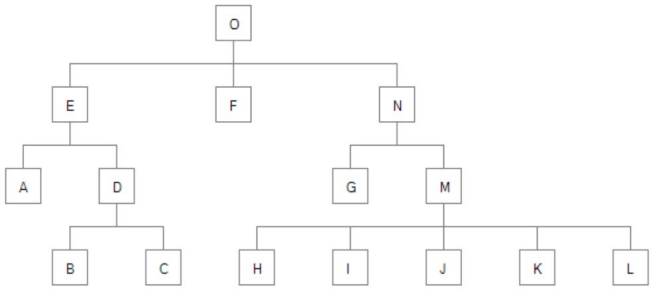
Layout：

Naïve recursive Layout:按照叶子的数量不断的划分子树空间

Reingold & Tiflord layout：空间的利用率会很高，密度最大，对称性很强，compact space.

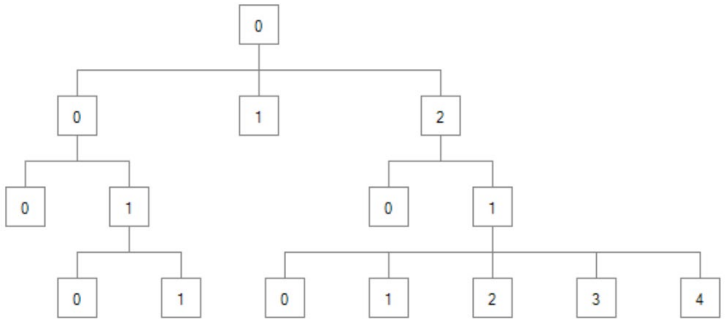
首先后序遍历到”shifts”,然后将右子树想左子树靠近，这样递归的不断进行下去，最后再来一遍前序遍历去设置 breadth dimension 的坐标。

详细步骤：假设要绘制的原始的 tree 的结构如下：

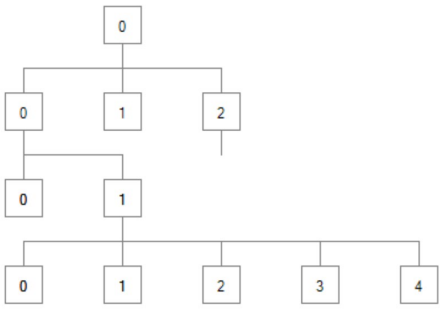


a) First Traversal of Tree:首先 run 一次 post-order traversal（后序遍历），意味着是从 bottom 向 up，从左到右来绘制 tree 的。原图中 A~G 就表示了后序遍历的顺序

b) Assigning initial X values：对每一个 node 都计算 local X 的值。这是仅仅只关于子节点的 X 值。赋值的规则是，如果该节点左边没有其兄弟节点，那么就赋值为 0，否则的话，就是其左兄弟节点的值加一。初始化赋值后如下图：



此时如果直接将初始的 X 值作为各个节点在 X axis 的坐标绘制，就会出现很多的节点之间相互 overlap（重叠），如下图所示。所以下一步要做的就是将子节点放置在其各自的父节点之下。



c) The mod property
剩余的步骤涉及到将 node 和其子节点位移很多次。为了能够这样做，通常必须遍历该子树中的所有节点并且增加其 X 的值。但这样的性能不好，尤其是那种很大的树。为了避免这个问题，我们使用 Mod property 去告诉 node，它应该将其子节点移动多远，之后我们将会第二次遍历（前序遍历）整个树去决定每个节点的最终的 X 的值。

去移动一个节点和其子节点在这次迭代中，增加节点的 X 的值和 Mod property 的值，增加的量随我们。

d) Positioning Child Nodes under parents
这一步是将子节点剧中置于父节点下。

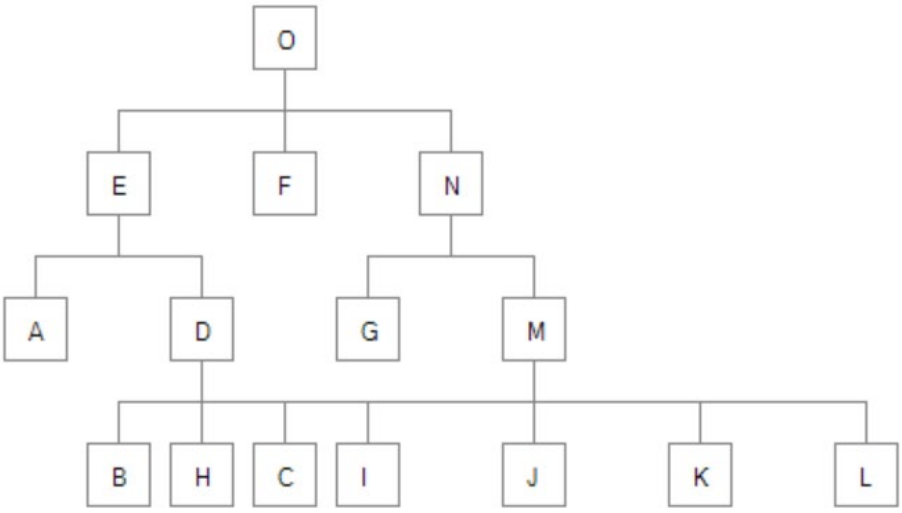
首先，找到一个 X 的值，使得该节点能够位于其子节点的中心。
1. 如果节点只有一个孩子，那么渴望的 X 值就是跟其子节点的 X 的值一样

2. 如果该节点有多余一个的孩子，那么渴望的 X 的值就是第一个孩子和最后一个孩子的中点的 X 的值。

接下来，检查该父亲节点的左边是否有其兄弟节点，可以通过检查该节点是否是其父节点的第一个节点。

1. 如果该节点是其父节点的第一个子节点，就将该节点的 X 的值设置为渴望的 X 的值，
2. 如果不是的话，就将该节点的 Mod 值改成 node.x - 渴望的 X 的值。来移动父节点下的子节点。

在第二次遍历结束后（计算了每个节点的 X 的值），会修正每个节点都在其父亲节点下，但这样没有阻止 overlapping。如图所示：B，C,H 和 I

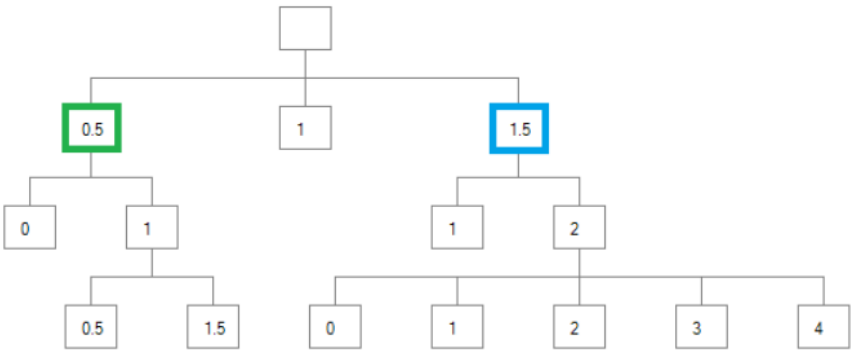


e) Checking for node Conflicts

基本的逻辑是，如果节点有孩子，我们就需要遍历它的所有层的孩子并且保证子节点的 X 的值不会和同一层的其他已经定位了的节点冲突。

所以，我们需要遍历当前的节点的所有的子节点，然后对每一个 Y 值我们都要记录最小的 X position。对于这个节点的每一个在其左边的兄弟节点，我们要遍历其有的子节点并且对于每一个 Y，都要记录其最大的 X position。这被称之为 Contour

举个例子：如下图。



如果蓝色的节点是当前的节点，那么它的左 contour 是{1.5,1,0}，即其每一层孩子的最小的 X 的值。对于其左边的兄弟节点来说（绿色），我们要记录的这个节点的所有的孩子节点中在每一层（每一个 Y）的最大值，即{0.5,1,1.5}

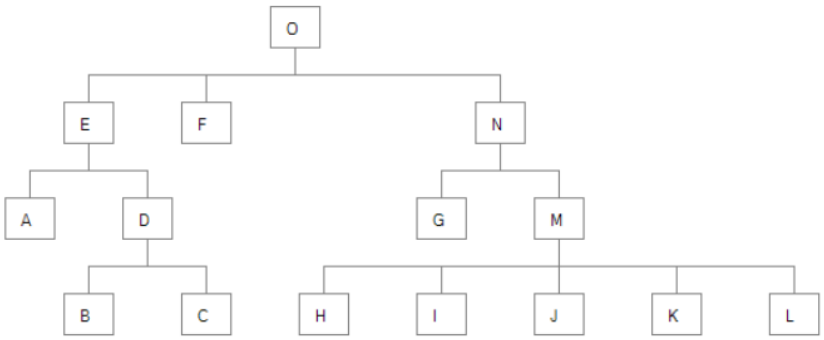
需要明白的是，这个 check 用的是 X 的迄今为止的最终的值，并不是一开始初始化时候用的值。

我们可以从上图看出来，第一层和第二层都有重叠，所以我们需要将蓝色的节点移的足够远，直到它的左 Contour 不再和绿色节点的右 Contour 矛盾

在这个情况下，最大的 overlapping distance 是 1.5，并且我们会额外的加一个 1，来防止某些节点会在某些节点的正上方。所欲最终要移动的距离是 2.5

f) Fixing middle tress afer resolving a confict

虽然前面的步骤解决了 overlapping，但是通过看下图可以发现，现在的 tree 的 layout 任然中部是有很大的空白，注意下图的 F 节点。

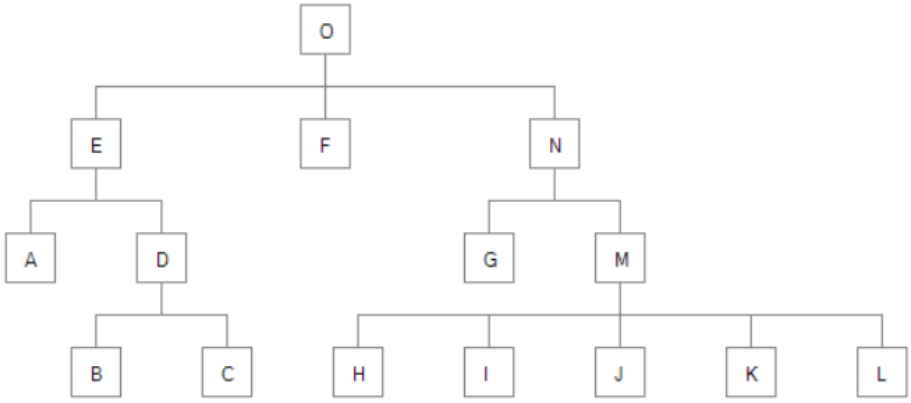


我们移动了 N 节点，是因为之前 N 节点及其所有的子节点会和 E 节点的所有子节点矛盾，但是我们也要平均地移动这两个矛盾节点之间的兄弟节点。

平均移动的距离是：之前 N 节点及其所有子节点移动的距离（2.5），除以：在矛盾节点啊之间的所有的子节点数目+1，在这个案例中就是只有 F 节点是处于矛盾节点之间的，所有 1+1 = 2，所以 F 节点应该移动的距离是（2.5）/2 = 1.25.

有一点需要明确的是：我们在移动了 F 节点后，并没有造成任何的矛盾，我们需要再一次的 check 一下矛盾。

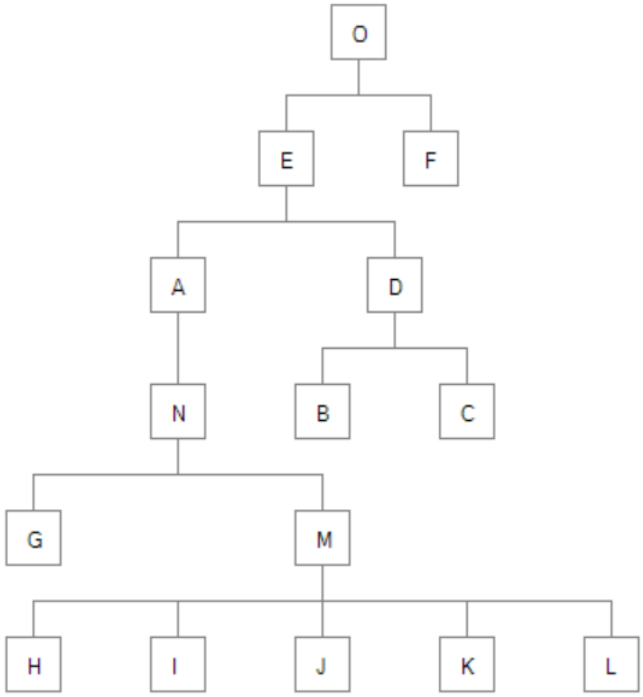
最终结果如下图：



g) Second Traversal of Tree

有些节点的 Mod 值可能是负数，这会把孩子节点的 X 值变成负数，并且当我们绘制该图像的时候，会出现部分他图像出现在屏幕的外部。比如，如果我们将 N 节点放在 A 节点的小面，而不是 O 节点。

我们可以在计算 Mod 值的时候就进行检查，确保其值不是负的。然而，这么做会仅仅只移动当前的节点，会造成一些不期望的空白出现，如下图，G 和 M 之间。

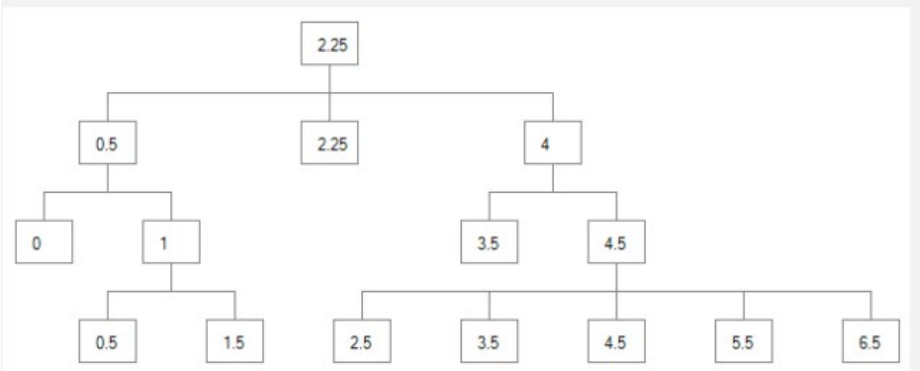


修复这个问题的最好的办法就是移动 root 节点足够的多，以至于没有任何节点的 X 的值会是负数。

为了能够这么做，我们可以先获取根节点的 Right Contour（右轮廓），找到最小的 X 的值，如果这个值是负数，我们就移动根节点 X 的绝对值的距离。要记住，当移动一个子树的时候，我们既要调整 X 的值，有调整 Mod 的值。

h) Third and Final Traversal

这一次遍历计算每个节点的最终的位置。计算的方法就是每个节点的 X 的值加上其父亲节点的 Mod 值。本案例中 X 的值就应该如下所示。

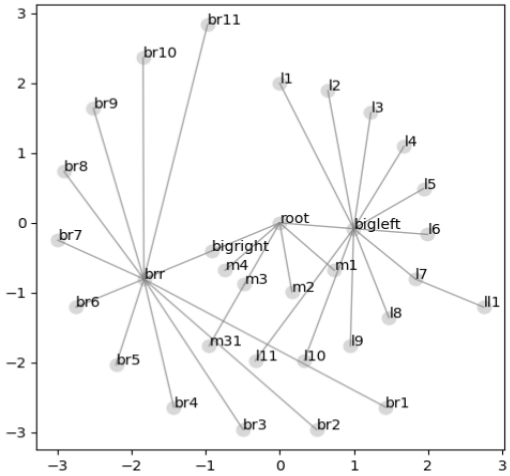


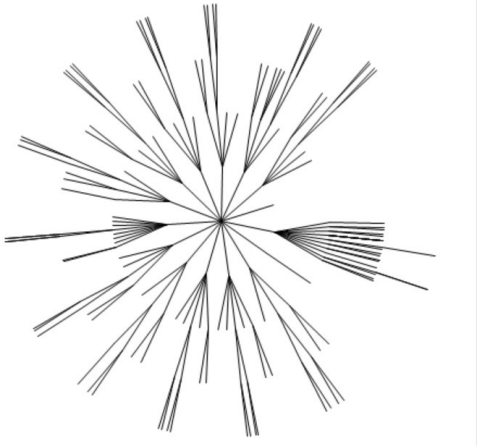
Reingold-Tilford algorithm 就是上述这样

Radial Tree：

使用的是极坐标系，极径代表了树的深度，而极点代表了树的根部，角度扇区分配给了子树，这样的结构是递归的布局。可以使用 RT algorithm 去布局

接下来使用 Reingold-Tilford 算法去实现 Radial Tree 的 layout。
手动的输入数据，看一下结果可以发现，radial tree 的结果





结论分析与体会：

附录：程序源代码