作业 3 二叉树问题

解雲暄 3190105871

索引

1 问题描述

问题提供了一个 MFC 工程 Demo，其功能包括：

- 根据输入序列建立二叉树，并输出其前序、中序、后序遍历；

在此基础上实现：

- 输出其层序遍历；
- 绘制二叉树；
- 镜像绘制二叉树（注：镜像绘制时不更新遍历序列）；
- 给出一个结点到另一个结点的最短路径。

2 算法与数据结构概述

在实现以上功能的过程中，值得说明的算法有 2 个部分：

1. 层序遍历的产生
2. 寻路

我们分别在 2.2 和 2.3 小节解释其算法设计。

2.1 数据结构概述

本工程的数据结构沿用了 Demo 中的二叉树类，在其基础上做了一些修改：

- 为了方便寻路，我们在二叉树结点类型 `struct Btnode` 中添加了 `father` 字段，指向其父结点。
- 为了方便其他计算，我们在二叉树类 `Binary_Tree` 中增加了
    - `int level` 字段表示二叉树的高度；
    - `getLevel()` 和 `getRoot()` 这两个 getter；
    - 增加了 `levelrav_Binary_Tree()` 函数实现层序遍历。

2.2 层序遍历

层序遍历的算法非常简单。维护一个二叉树结点队列，首先将树的根结点入队。当队列不为空时，每次取出队头，并将其两个子结点（如果非空）入队。循环至队列为空时层序遍历即结束。

在本工程中，层序遍历的另一个使命是统计二叉树的深度（定义空树的深度为 ）。我们维护另一个数字队列，首先入队数字 1 表示根结点所在层数为 1。数字队列的出队入队与结点队列同步；每次将子结点入队时在数字队列入队当前结点深度（即数字队列队头） + 1。这样出现过的最大的数字（即最后一个数字）就是二叉树的深度了。

层序遍历在 `levelrav()` 函数实现。

2.3 寻路

寻路操作由函数 `CString findRoute(Btnode<TCHAR> *tree, TCHAR from, TCHAR to)` 实现。`tree` 是树根结点的指针，`from` 和 `to` 是寻路的起点和终点。这个函数的流程是：

- 调用 `Btnode<TCHAR> *findNode(Btnode<TCHAR> *tree, TCHAR ch)` 函数寻找起点和终点对应的树节点指针；
    - 该函数通过遍历整棵树查找对应结点，同时维护每个结点的 `father` 字段；细节不表。
- 如果任一结点没有找到，返回字符串 `L"Point given not found."`；
- 用两个 `vector` 记录两个结点到根结点的路径；
- 显然，这两个 `vector` 的尾部 1 个或多个结点是相同的（至少根结点是相同的），我们找到最靠前的那个相同结点就能找到这条最短的路径（也就是唯一的简单路径）了。具体的实现参见源代码。

3 MFC 设计概述

3.1 绘图参数设计

本工程的主体设计在于二叉树的绘制。我们希望根据二叉树的实际大小来调整绘图的比例尺；即，在绘图区域大小一定的情况下，二叉树越深，每个结点就画得越小。

记绘图控件的宽度、高度分别为 W, H，二叉树的总高度为 L；用 R 表示单结点的半径；用 (i, j) 表示需要画的结点在二叉树中的位置，其中 i 表示结点所在的层数或深度（根结点为 1），$j < 2^i$ 表示其在这一层的第几个结点（按满二叉树计算）；(x, y) 表示该结点实际所画圆形的外接正方形的左上顶点的坐标。

我们设计如下计算公式，实现较好的结点大小和比例关系：

$$R = \min(\frac{W}{2^L - 1}, \frac{H}{2L}) \tag{1}$$

$$x = \begin{cases} 0, & L = 1 \\ (2^{L-i} + (j-1) \cdot 2^{L-i+1} - 1)R, & L > 1 \end{cases} \tag{2}$$

$$y = \begin{cases} 0, & L = 1 \\ (i-1) \cdot \dfrac{H - 2R}{L - 1}, & L > 1 \end{cases} \tag{3}$$

这样的设计可以实现的效果是，最下面一层的 2^L 个结点平分控件宽度，父结点的横向坐标是其两个子结点的均值；每层结点之间的纵向距离一致。

在绘制每一个结点时，结点还会得知其父结点的 (x, y) 便于进行连线。我们记当前结点的圆心为 C_2，父结点的圆心为 C_1。我们希望在两个圆心之间连一条线，但是我们希望这条线不会覆盖所绘制的结点；因此我们作一定的偏移：

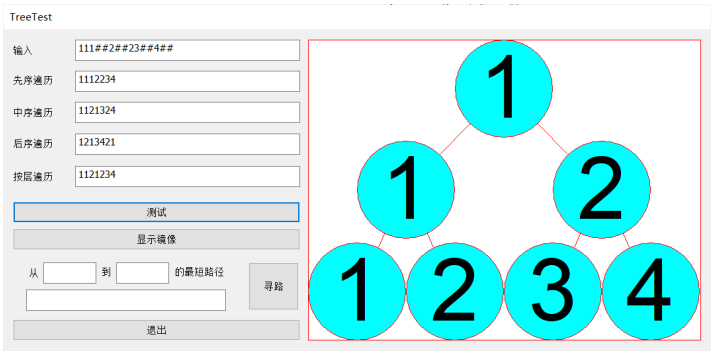
$$\begin{cases} \Delta x = \dfrac{R}{dis} \cdot (x_2 - x_1) \\ \Delta y = \dfrac{R}{dis} \cdot (y_2 - y_1) \end{cases} \tag{4}$$

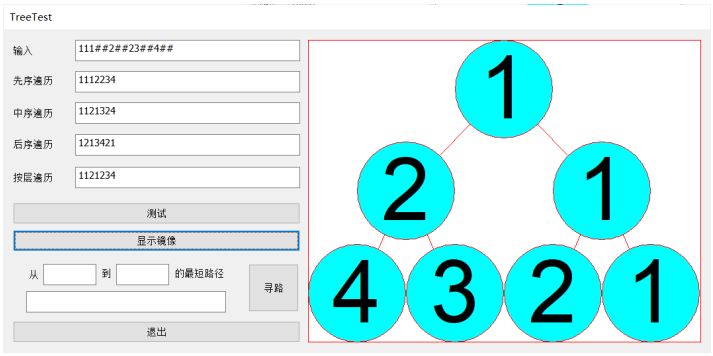
其中 dis 是两个圆心之间的距离。

3.2 镜像的处理

讨论镜像带来的序号影响。

我们在 3.1 计算绘图位置时的重要参数是 (i, j)。镜像并不改变 i，我们研究其对 j 的变化：





可以看到，在镜像之前，每个结点的左结点的 $j=2j_0-1$，而右节点为 $2j_0$；其中 j_0 是其父亲节点的 j。而镜像后，这一关系发生了翻转。因此我们可以仅仅通过改变 j 来实现镜像。

3.3 绘图的代码实现

按钮的响应函数可以调用 `void CTreeTestDlg::DrawTree(bool isInverse)` 函数来启动绘图。其中参数 `isInverse` 为 `true` 表示绘制的是镜像图；否则不是。

该函数首先对控件句柄、画笔、画刷进行一些配置，然后通过在绘图控件画一个白色矩形实现清屏。该函数根据控件的大小参数，按照 3.1 节所述的方式计算出半径 `radius`，并将字体大小设置为 `2 * radius`，设置文字背景透明。该函数调用 `DrawTreeNode()` 函数实现结点的绘制。

函数 `DrawTreeNode()` 以递归形式进行绘制，每次绘制一个结点。其声明格式为：

```
void DrawTreeNode(HDC &hdc, Btnode<TCHAR> *tree, bool isInverse,
                  int width, int height, int radius, int totLev,
                  int lev, int ind, int lastX0, int lastY0)
```

其中，`hdc` 是绘图句柄，`tree` 是树的一个结点的指针，其他参数的含义是明显的。

这个函数依次完成如下工作：

1. 计算 (x,y)；

2. 绘制表示结点的圆：`Ellipse(hdc, x0, y0, x0 + 2 * radius, y0 + 2 * radius);`
3. 绘制结点文字：`TextOutW(hdc, x0 + radius / 2, y0, s.GetBuffer(0), s.GetLength());`
4. 画父节点与该结点之间的连线；
5. 递归调用该函数，实现对子结点的绘制：

```
if (tree->lchild)
    DrawTreeNode(hdc, tree->lchild, isInverse, width, height,
                 radius, totLev, lev + 1,
                 !isInverse ? ind * 2 - 1 : ind * 2, x0, y0);
if (tree->rchild)
    DrawTreeNode(hdc, tree->rchild, isInverse, width, height,
                 radius, totLev, lev + 1,
                 !isInverse ? ind * 2 : ind * 2 - 1, x0, y0);
```

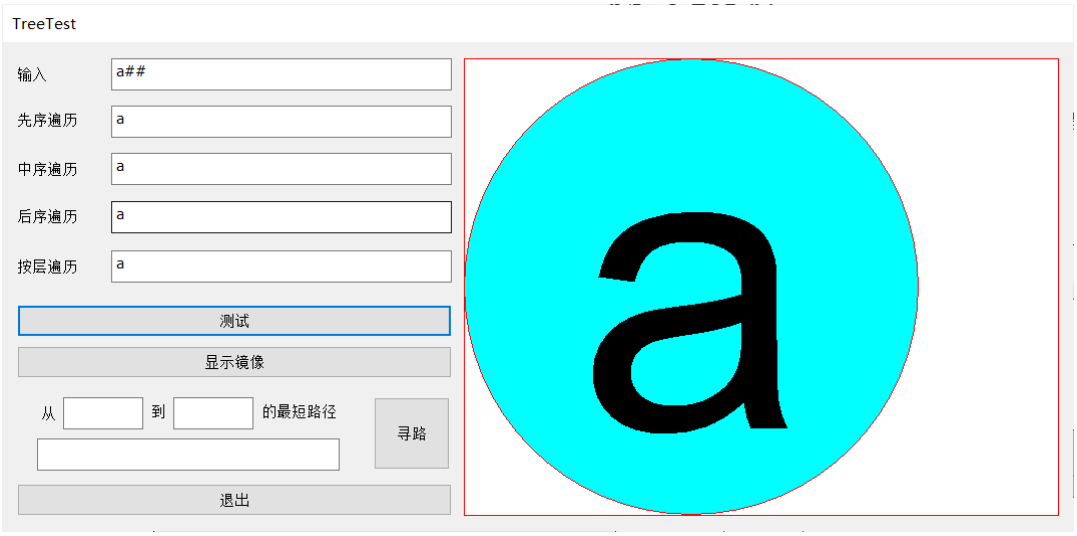
可以看到，这里根据 `isInverse` 的取值，按照 3.2 节中的讨论结果为子结点传入不同的 `j`。

4 测试与分析

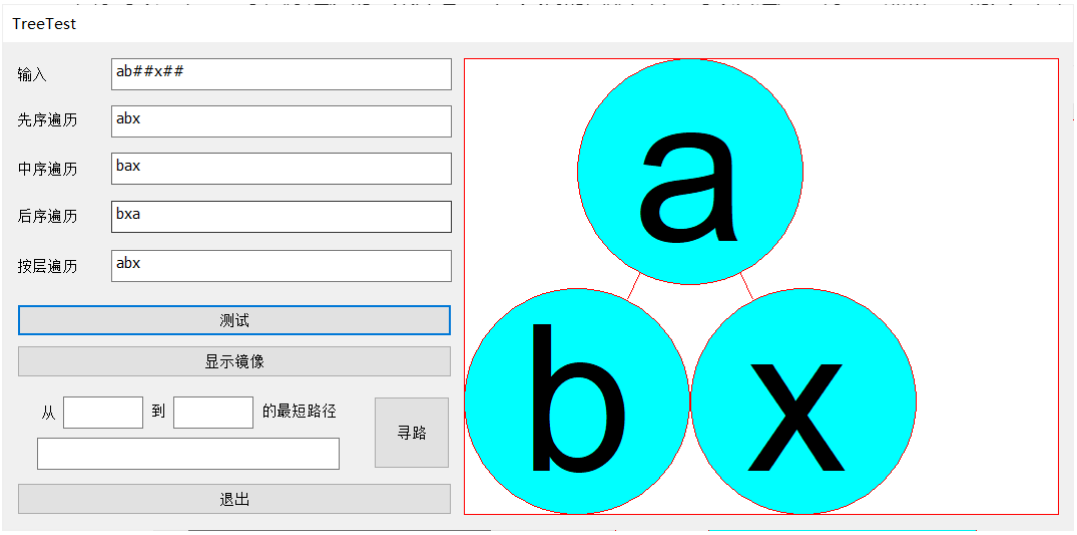
4.1 绘图和层序遍历测试

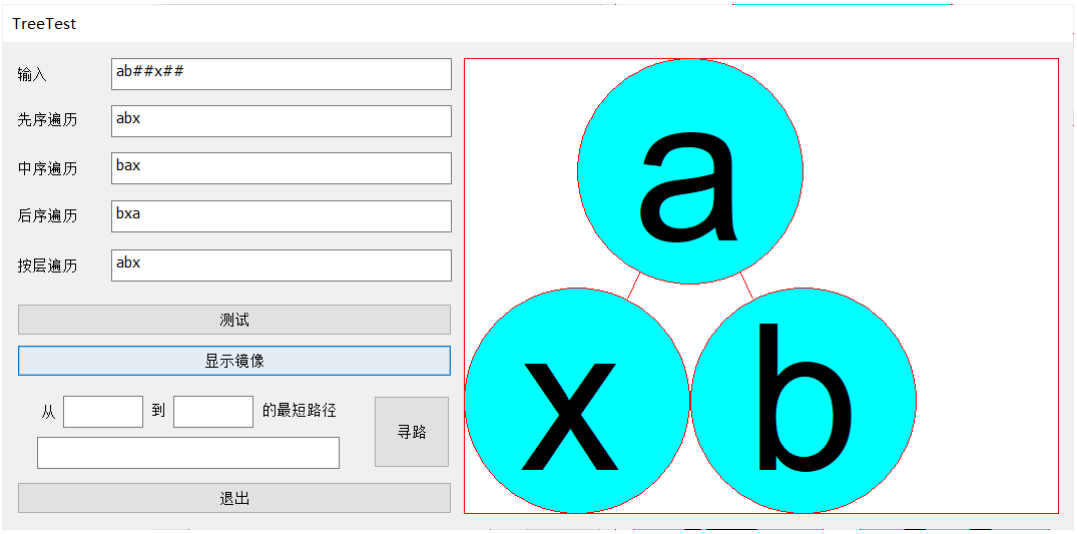
程序可以正确地显示层序遍历的结果并绘制出二叉树的图形，并且可以自适应地调整结点和文字的大小；程序也可以正确地绘制二叉树的镜像图形：

一个结点

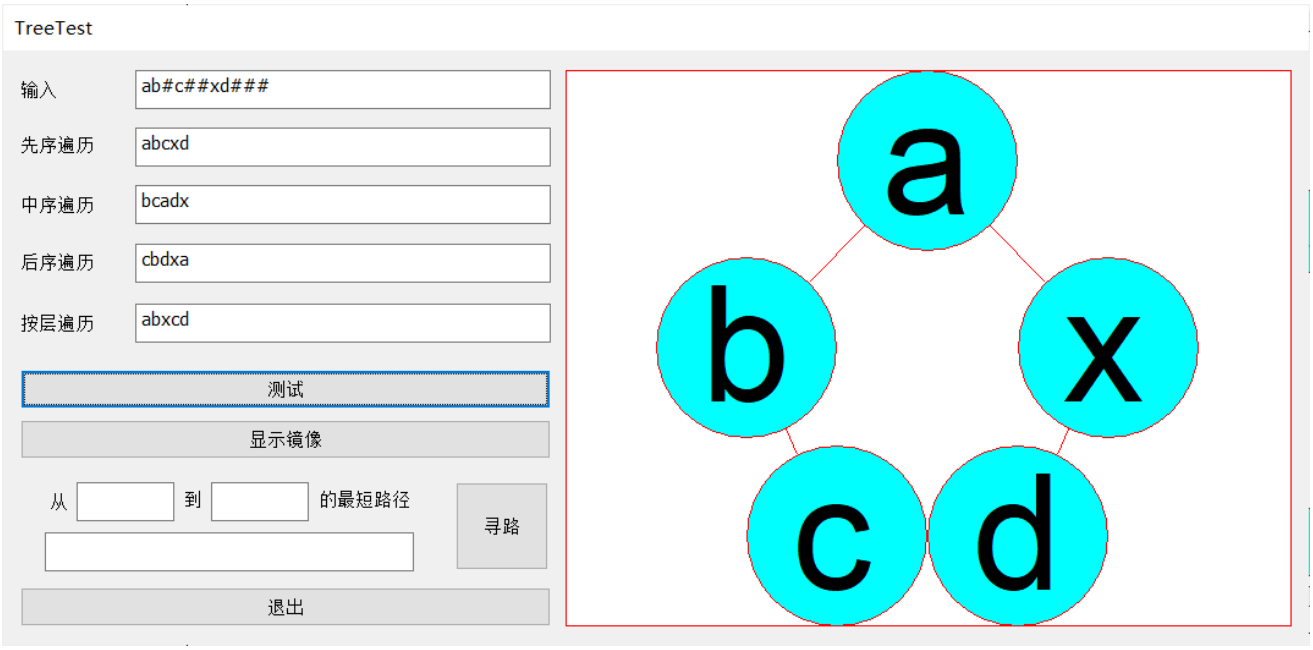


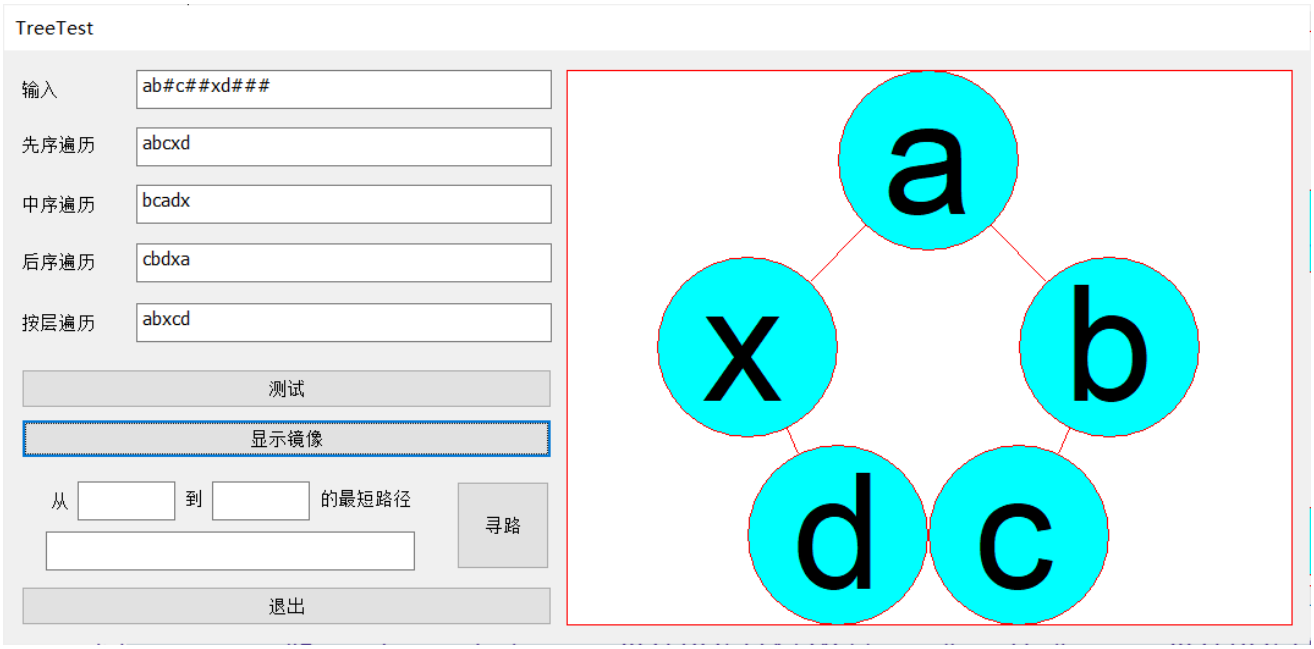
两层及其镜像



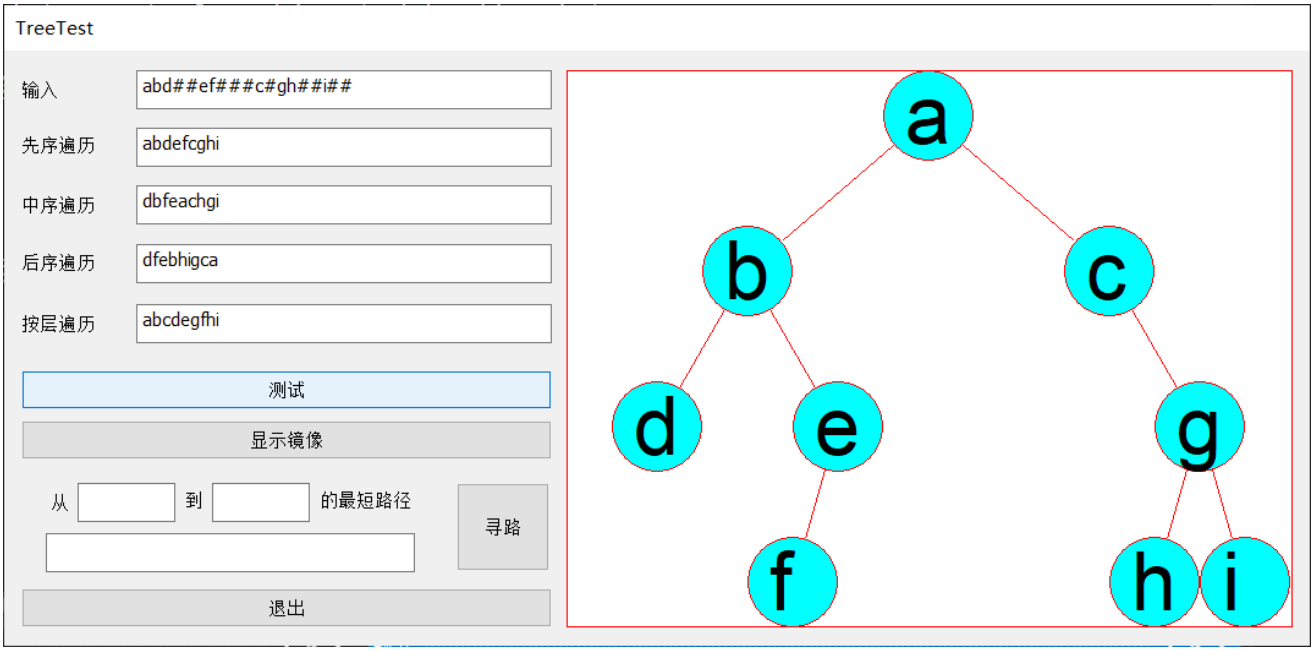


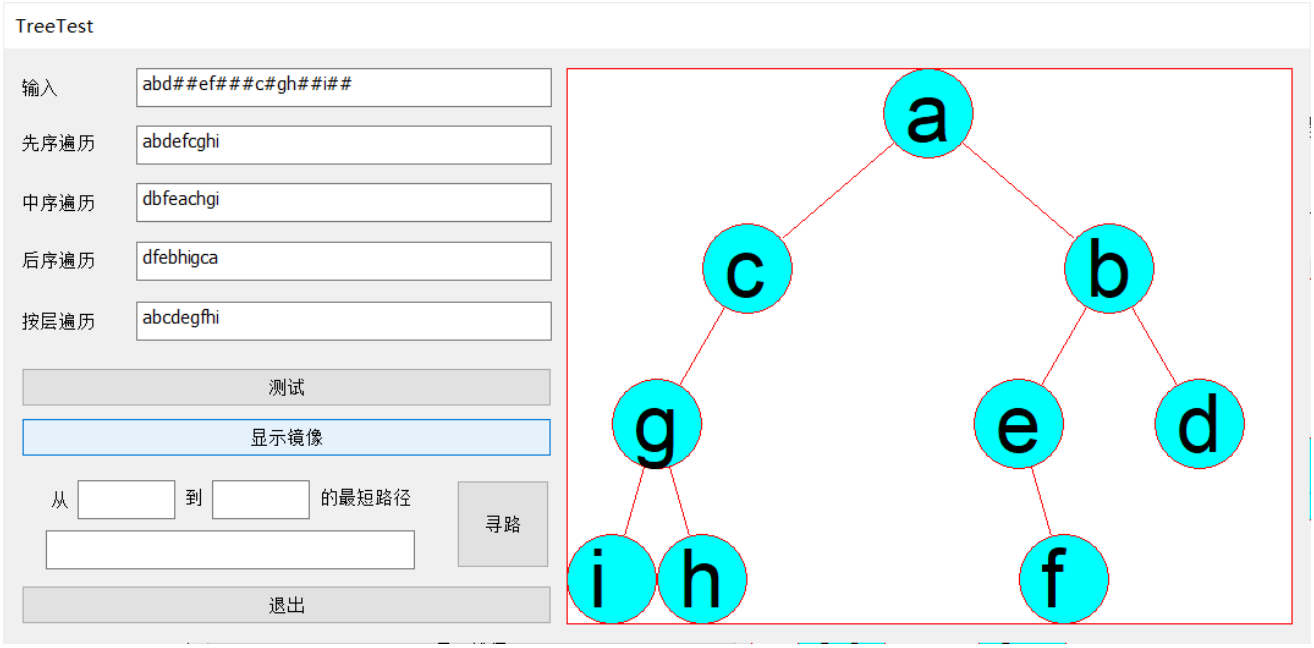
三层及其镜像




TreeTest
输入
ab#c##xd###
先序遍历
abcxd
中序遍历
bcadx
后序遍历
cbdxa
按层遍历
abxcd
测试
显示镜像
从
到
的最短路径
寻路
退出
a
x
b
d
c

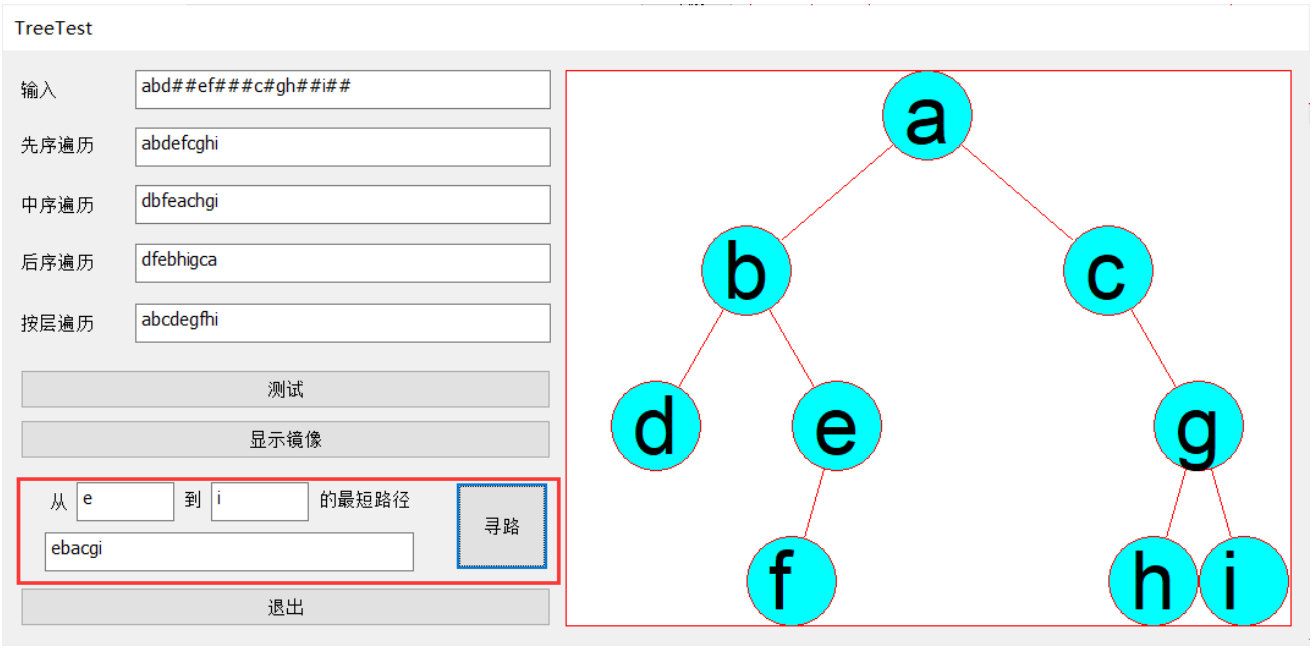
四层及其镜像


TreeTest
输入
abd##ef###c#gh###i##
先序遍历
abdefcghi
中序遍历
dbfeachgi
后序遍历
dfebhigca
按层遍历
abcdegfhi
测试
显示镜像
从
到
的最短路径
寻路
退出
a
b
c
d
e
g
f
h
i

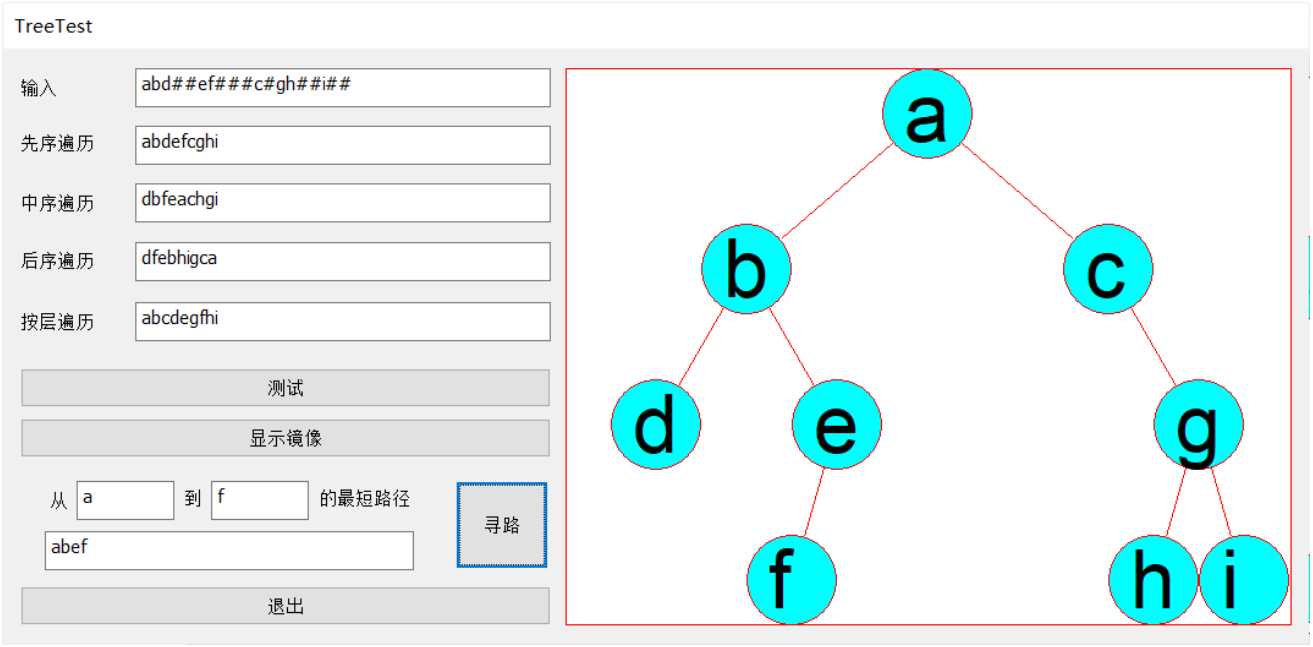

TreeTest
输入
abd##ef###c#gh###i##
先序遍历
abdefcghi
中序遍历
dbfeachgi
后序遍历
dfebhigca
按层遍历
abcdegfhi
测试
显示镜像
从
到
的最短路径
寻路
退出
a
c
b
g
e
d
i
h
f

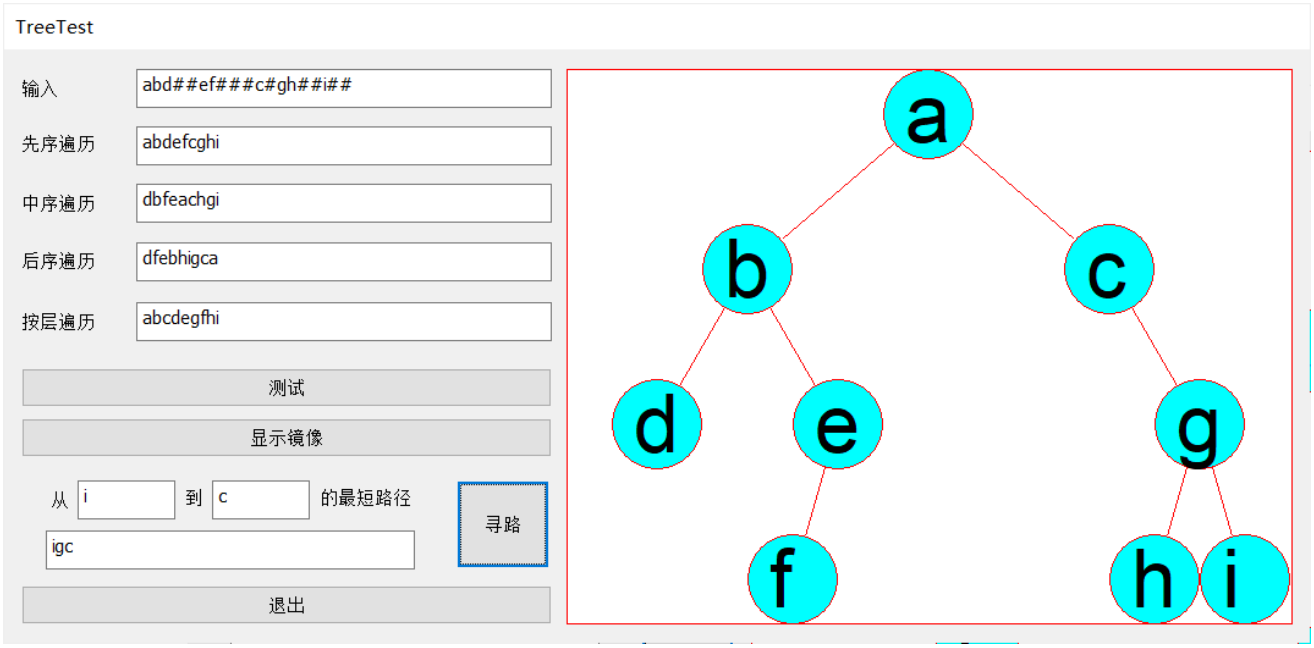
4.2 寻路测试

一般情况

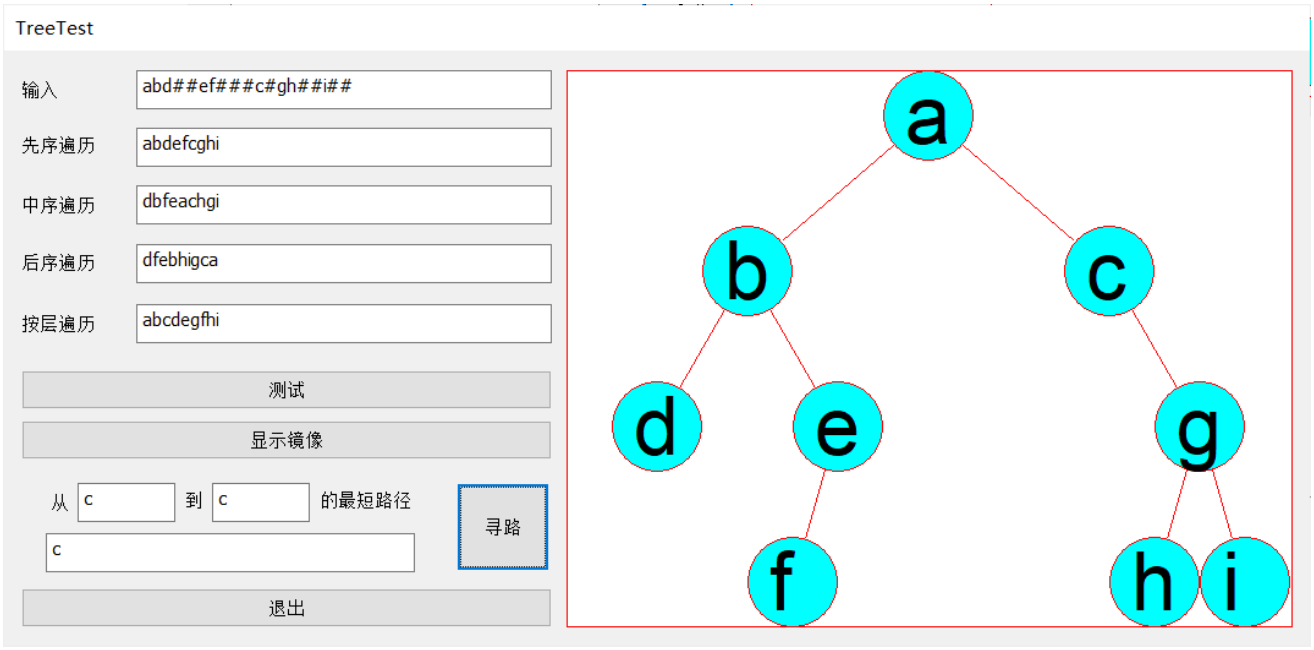


一个点为另一个点的祖先的情况

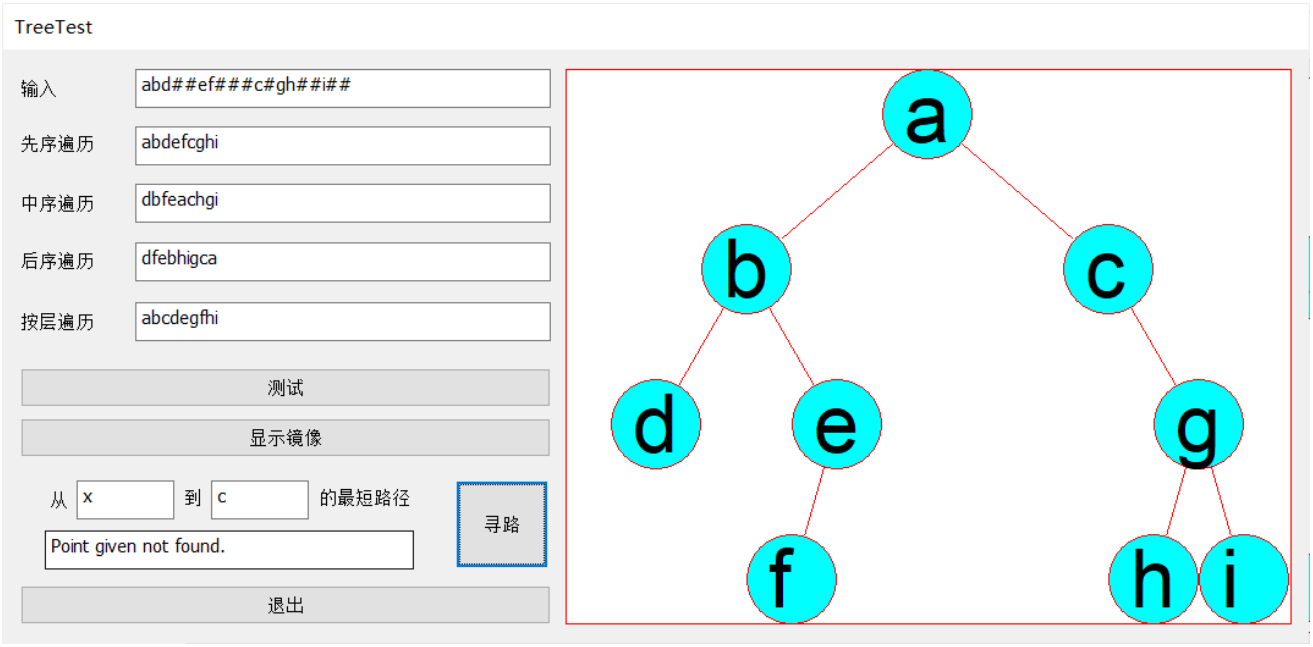



TreeTest
输入
abd##ef###c#gh##i##
先序遍历
abdefcghi
中序遍历
dbfeachgi
后序遍历
dfebhigca
按层遍历
abcdegfhi
测试
显示镜像
从
i
到
c
的最短路径
igc
寻路
退出
a
b
c
d
e
g
f
h
i

同一个点


TreeTest
输入
abd##ef###c#gh##i##
先序遍历
abdefcghi
中序遍历
dbfeachgi
后序遍历
dfebhigca
按层遍历
abcdegfhi
测试
显示镜像
从
c
到
c
的最短路径
c
寻路
退出
a
b
c
d
e
g
f
h
i

输入有误



4.3 分析

本工程在完成了所有要求的功能以上，**精心计算了结点的绘制位置**。绘制前，程序会根据二叉树的层数计算出一个适合的结点直径和字体大小，并给出公式进一步计算每个结点的显示位置以及绘制线条的位置；保证显示结果的完整和美观。

同时本工程也存在一些问题。例如，建树序列的输入没有错误检查；这是因为没有进一步对 Demo 做升级。另外，由于绘制位置的坐标是用 `int` 表示的，而且计算过程中存在很多除法，其精度会有一定损失，因此绘图结果（尤其是文字显示）会略偏离中心等。

4.4 感想

MFC 可以说是一个略有“过时”的框架，其接口定义也并不是那么的完整、合理和规范。因此希望解决一些问题（如清空绘图区等）时很难搜索到有效的解决方案。虽然这也有助于促进自行解决问题的相关思考（比如这里我们通过绘制一个覆盖控件的白色矩形实现视觉上的覆盖清空），但是这也降低了开发过程中的体验和效率。如果可以考虑升级为一些相对流行且文档完整的应用框架，可能更有助于将同学们的精力专注到解决问题本身上来。