ECE368 Programming Assignment (PA#1)
Due on July 2nd & 6th, 2021 11:59 pm

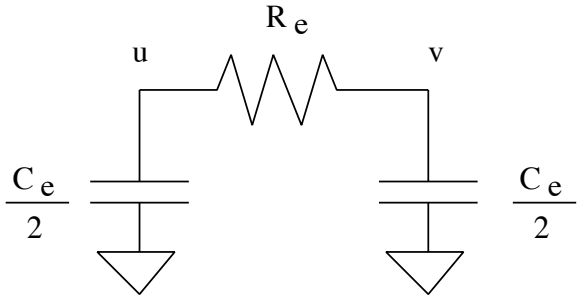
Description

You are required to implement a program to compute the signal delay of an RC (resistive-capacitive) tree, represented by a binary tree.

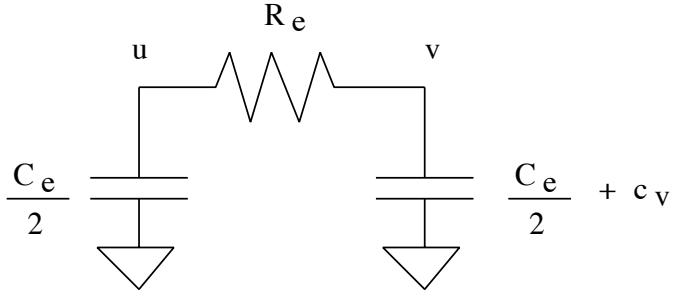
Every tree edge in the binary tree is a wire with some resistance and some capacitance. Consider a tree edge $e = (u, v)$ of length l_e connecting a parent node u and to a child node v. The resistance and capacitance of the wire are given by

$$R_e = r \times l_e, \quad C_e = c \times l_e,$$

where r and c are per-unit-length resistance and per-unit-length capacitance, respectively. A first-order model of the wire is given below:



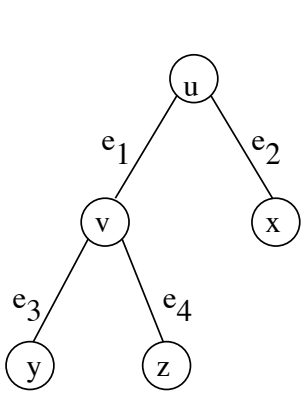
If the child node v happens to be a leaf node, it has an additional capacitance value associated with it, called the sink capacitance, denoted by c_v. The corresponding circuit is given below:



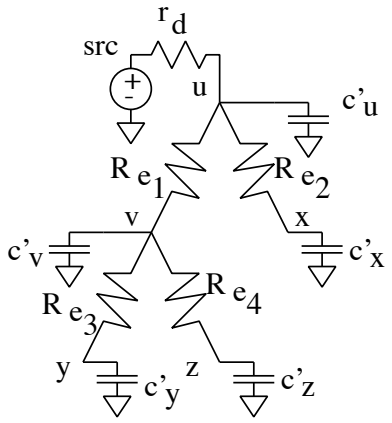
The tree is driven at its root by a driver (or buffer) with a output resistance r_d. Therefore, the corresponding RC-tree of a binary tree is as follows:

In the preceding figure, c'_i corresponds to the total capacitance at node i. Therefore,

$$c'_u = \frac{C_{e_1}}{2} + \frac{C_{e_2}}{2},$$



(a) A binary tree



(b) The corresponding RC tree

$$
\begin{aligned}
c'_v &= \frac{C_{e_1}}{2} + \frac{C_{e_3}}{2} + \frac{C_{e_4}}{2}, \\
c'_x &= \frac{C_{e_2}}{2} + c_x, \\
c'_y &= \frac{C_{e_3}}{2} + c_y, \\
c'_z &= \frac{C_{e_4}}{2} + c_z.
\end{aligned}
$$

Moreover, the driver output resistance r_d is between a source node, denoted as *src* and the root of the binary tree u. We can think of r_d being the resistance of the edge (src, u).

Given an RC tree T, the following delay model gives an approximation of the delay from the source node *src* to node j. Let $\text{Path}(src, j)$ denote the path in the tree from node *src* to node j. Moreover, let $R_{\text{Path}(src,i)\cap\text{Path}(src,j)}$ denote the total resistance along the path common to $\text{Path}(src, i)$ and $\text{Path}(src, j)$. The delay from *src* to j is given by the following expression:

$$
t_j = \frac{1}{R_{\text{Path}(src,j)}} \sum_{i \in T} c'_i \times R^2_{\text{Path}(src,i)\cap\text{Path}(src,j)}.
$$

Under this model, the delays of nodes v and z, for example, are:

$$
\begin{aligned}
t_v &= \frac{1}{r_d + R_{e_1}} \left\{ c'_u r_d^2 + c'_v (r_d + R_{e_1})^2 + c'_x r_d^2 + c'_y (r_d + R_{e_1})^2 + c'_z (r_d + R_{e_1})^2 \right\}, \\
t_z &= \frac{1}{r_d + R_{e_1} + R_{e_4}} \left\{ c'_u r_d^2 + c'_v (r_d + R_{e_1})^2 + c'_x r_d^2 + c'_y (r_d + R_{e_1})^2 + c'_z (r_d + R_{e_1} + R_{e_4})^2 \right\}.
\end{aligned}
$$

This delay model has an unusual property. The delays of nodes along the path from the source to a sink do not increase monotonically. For example, it is possible that t_v is less than t_z even though node z appears at the end of the path from *src* to z, and node v appears in the middle of the path.

For the delay model to be properly defined for all nodes in a tree, you may assume that the resistance r_d is a strictly positive value.

Using an appropriate tree-traversal algorithm, the delay of a single node can be computed in $O(n)$ time-complexity, where n is the total number of nodes in the RC tree. In fact, the delays of all nodes can be computed in $O(n)$ time-complexity. In order to achieve that, it is important to analyze how t_z can be computed from t_v in the preceding example.

Deliverables

In this assignment, you are required to develop your own include file(s) and source file(s), which can be compiled with the following command:

```
gcc -std=c99 -pedantic -Wvla -Wall -Wshadow -O3 *.c -o pa1
```

It is recommended that while you are developing your program, you use the "`-g`" flag instead of the "`-O3`" flag for compilation so that you can use a debugger if necessary. All declarations and definition of data types and the functions associated with the data types should reside in the include file `delay.h` or source file `delay.c`. The main function should reside in the file `pa1.c`

The executable `pa1` would be invoked as follows:

```
./pa1 input_filename output_filename1 output_filename2
```

The executable loads the RC tree from a file whose filename is given as `input_filename` (note that the `input_filename` can be any arbitrary valid filename; it is effectively the string stored as `argv[1]`), prints the topology of the RC tree in pre-order to a file whose filename is given as `output_filename1` (effectively, `argv[2]`), calculates the delay of all nodes in the tree, and print to a file whose filename is given as `output_filename2` (effectively, `argv[3]`) the delays of all leaf nodes.

Format of input file

The format of the input file specified by the `input_filename` is as follows. The first line of the file contains three numbers (in the format of `"%le %le %le\n"`), where the first `double` specifies the output resistance at the source, r_d (Ω), the second `double` specifies the per-unit-length resistance of a wire, r (Ω/unit-length), and the third `double` specifies the per-unit-length capacitance of a wire, c (F/unit-length).

The rest of the file is divided into lines, and each line corresponds to a node in the binary tree. The order in which the nodes are printed is based on a post-order traversal of the binary tree. If it is a leaf node (which is a sink), it has been printed with the format `"%d(%le)\n"`, where the `int` is the label of the sink, and the `double` is the sink node capacitance (F). If it is a non-leaf node, it has been printed with the format `"(%le %le)\n"`, where the first `double` is the wire length to the left child and the second `double` is the wire length to the right child.

Format of first output file

The first output file is a pre-order printing of the given RC tree. Each line corresponds to a node in the binary tree. If it is a leaf node (which is a sink), you print the node with the format

"%d(%le)\n", where the int is the label of the sink, and the double is the sink node capacitance (F). If it is a non-leaf node, you print with the format "(%le %le)\n", where the first double is the wire length to the left child and the second double is the wire length to the right child.

Format of second output file

The second output file contains a pre-order printing of the delays of the leaf nodes of the given RC tree in the binary format. For each leaf node, you write the label (int) and the delay (double). The file size of the second output file should be the number of leaf nodes multipled by the sum of sizeof(int) and sizeof(double).

Electronic Submission

The project requires the submission (electronically) of the C-code (source and include files) through *BrightSpace* . You should create and submit a zip file called pa1.zip which contains the .h and .c files. Your zip file should not contain a folder.

```
zip pa1.zip *.c *.h
```

You should submit pa1.zip to *BrightSpace* .

If you want to use a makefile for your assignment, please include the makefile in the zip file. If the zip file that you submit contains a makefile, we use that file to make your executable (by typing make at the command line to create the executable called pa1).

Grading

The assignment will be graded based on the two output files produced by your program, evaluated using sample files that are randomly generated. The first output accounts for 40 points and the second output accounts for 60 points. It is important that your program can accept any legitimate filenames as input or output files. Even if you cannot produce the second output file correctly, you should still write the main function such that it produces a valid first output file and an empty second output file so that you can earn partial credit.

It is important all the files that have been opened are closed and all the memory that have been allocated are freed before the program exits. Memory errors or any errors reported by valgrind will result in a 40-point penalty.

What you are given

You are given 5 sample input files (3.txt, 5.txt, 10.txt 100.txt, 1000.txt) and two sets of sample output files (3.pre, 3.delay, 5.pre, and 5.delay). The file 3.txt corresponds to the 3-sink example used in this document, with leaf nodes x, y, and z being labeled as 1, 2, and 3, respectively. The file 3.pre corresponds to the pre-order printing of this example, and 3.delay corresponds to the pre-order printing of the delays of sink nodes of this example. Similarly, 5.pre and 5.delay are the pre-order printings of all nodes and of all sink delays of the example in 5.txt, respectively.

To help you get started, we also provide you 3.delay.txt and 5.delay.txt, which are the labels and delays of sink nodes (in pre-order) in text format. In these files, each line corresponds to a sink and its delay. Each line is printed with the format "%d(%le)\n", where the int is the node label, and the double is the node delay. (While developing your program, you probably want to print the

second output file in this manner. When you are confident of the correctness of your program, you would then write to a binary file the node labels and delays.)

10.txt, 100.txt, and 1000.txt contain examples of 10 sinks, 100 sinks, and 1000 sinks, respectively. Note that you should not assume that you can deduce the size of the RC tree based on the filename.

Additional information

As this assignment involves floating point computation, it is unlikely that your results will match my results exactly. We will allow for some minor discrepancies between your results and my results.

Check out the *BrightSpace* website for any updates to these instructions.