ECE368 Programming Assignment #3

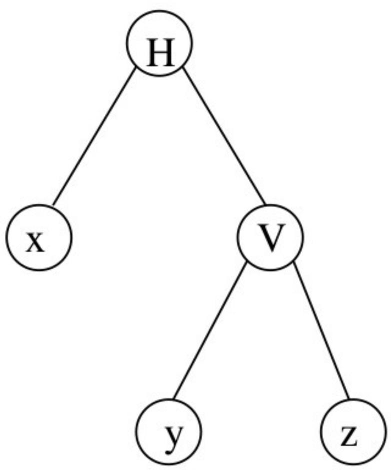
Due Wednesday, July 19, 2017, 11:59pm

Description:

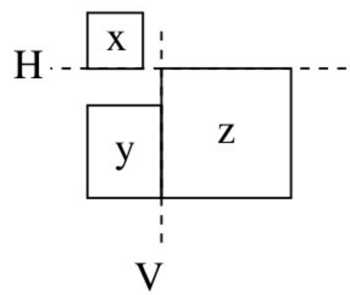
This project is to be completed on your own. You will implement a program involving tree traversal(s) to compute the “2D packing” of rectangles, represented by a binary tree.

In this binary tree, each leaf node represents a rectangle. Each internal node of the binary tree represents a partitioning of two groups of rectangles by a horizontal cutline or a vertical cutline. Let xHy (xV y) denote a (sub)tree, whose root node is a horizontal cut H (a vertical cut V). The left and right subtrees of H (V) are x and y, respectively. Assume that xHy means x is on top of and y is below the horizontal cut, and xV y means x is to the left and y is to the right of the vertical cut.

In the following figure, we show an example of a “packing” of three rectangles based on a given binary tree representation. Here, each subtree is enclosed by a smallest rectangular room. Assume that the dimensions (width, height) of the three rectangles x, y, and z are(3 , 3), (4 , 5), and (7 , 7). The smallest room containing the subtree yV z is of dimensions(11 , 7).



(a) A binary tree



(b) The corresponding packing

The smallest room containing the subtree xH (yV z) is of dimensions (11 , 10). This room is partitioned in to two smaller rooms: The top room is of dimensions (11 , 3) and it contains rectangle x, whereas the bottom room is of dimensions (11 , 7) and it contains rectangles y and z. We place the lower left corner of each rectangle to the lower left corner of its room.

Assuming that the lower left corner of the smallest room containing all the blocks is at coordinates (0 , 0), the coordinates of lower left corners of the three rectangles x, y, and z are respectively (0 , 7), (0 , 0) and (4 , 0). Note that even though there is space above y to accommodate x, x has to stay above the horizontal cutline in the “packing,” as shown in the figure.

Given a binary tree representation, the smallest rectangular room to enclose all rectangles and the coordinates of these rectangles in the corresponding packing can be computed in O(n) time complexity using appropriate tree-traversal algorithm(s).

Deliverables:

In this project, you are to develop your own include file **packing.h** that defines the structures you want to use and to declare the functions you need to manipulate the structures. You should define these functions in the source file **packing.c**. These functions should be called by a main function that resides in **packing_main.c**

Your programs should be compiled with the following command:

gcc -Werror -Wall -Wshadow -O3 packing.c packing_main.c -o proj3

The executable proj3 would be invoked as follows:

./proj3 input_file output_file

The executable loads the binary tree from input_file , performs packing, and saves the packing into output_file.

Your packing.c file should contain the following three functions (you may also need other functions):

1. Load_binary_tree_from_file

The input filename (including path) is provided in the command line. The binary tree contained in the file should be read in, parsed, and stored in the tree data structures defined by you. The input file is divided into lines, and each line corresponds to a node in the binary tree. If it is a leaf node (which is a real rectangle), it has been printed with the format "%d(%le, %le)\n", where the int is the label of the rectangle, the first double is the width of the rectangle, and the second double is the height of the rectangle.

If there are n rectangles in the packing, the labels are from 1 through n.

If it is non-leaf node, it is simply a character (followed by a newline character). The character is either ’V’ or ’H’ , representing either a vertical cutline or a horizontal cutline, respectively.

These nodes are printed in a postorder traversal of the binary tree. You would probably need to make use of a stack to help you with the reconstruction of the binary tree based on the input file.

2. Perform packing

Perform packing on the binary tree you have loaded in, using data type double for your computation of dimensions and coordinates. Note that you do not really pack the rectangles tightly, as shown in Figure (b).

3. Save packing to file

The output filename (including path) is provided in the command line. The file should contain
a line for each rectangle. The ordering of the rectangles in the output file should be the same as the
ordering of rectangles in the input file. Every line contains five fields:

- label (int), which specifies the label of the rectangle.
- width (double), the width of the rectangle.
- height (double), the height of the rectangle.
- xcoord(double), the x-coordinate of the rectangle.
- ycoord(double), the y-coordinate of the rectangle.

Your main function in the packing main.c file should call the aforementioned three functions
in the correct order. The program should report the following to the standard output:

Width: AAAA
Height: BBBB
X-coordinate: CCCC
Y-coordinate: DDDD

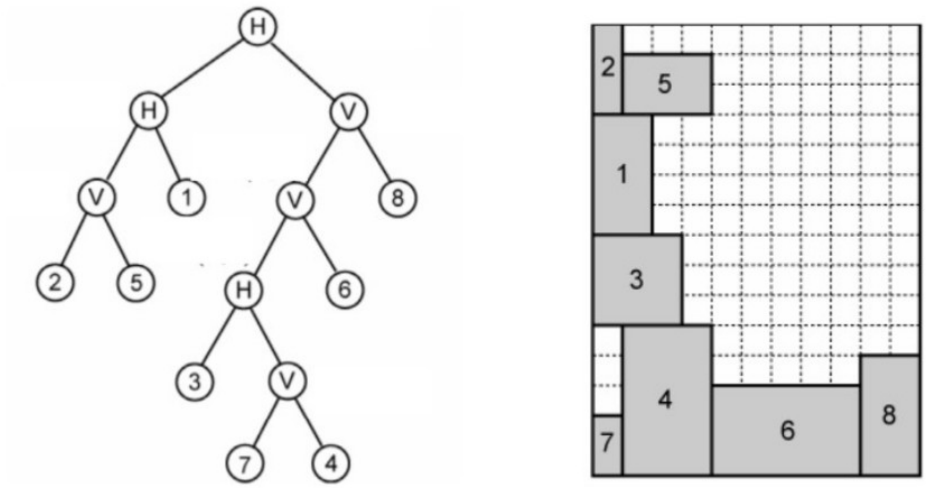
Here, AAAA, BBBB, CCCC, DDDD should be printed with the "%le" format. For Width and
Height , you should report the width and height of the smallest room that encloses the entire
packing specified by the binary tree. For X-coordinate and Y-coordinate, you should report
the coordinates of the rectangle with the largest node number.

Consider the following input file:

```
2(1.000000e+00,3.000000e+00)
5(3.000000e+00,2.000000e+00)
V
1(2.000000e+00,4.000000e+00)
H
3(3.000000e+00,3.000000e+00)
7(1.000000e+00,2.000000e+00)
4(3.000000e+00,5.000000e+00)
V
H
6(5.000000e+00,3.000000e+00)
V
8(2.000000e+00,4.000000e+00)
V
H
```

The following figure shows the corresponding binary tree and packing. The numbers in parentheses next to each internal node (i.e., non-leaf node) denote the width and height of the smallest room enclosing the rectangles in the subtree of this node. Therefore, the width and the height of the smallest room enclosing all rectangles are 11 and 15. The coordinates of the largest indexed node are (9 , 0).

The output file should have the following format:




```
2 1.000000e+00 3.000000e+00 0.000000e+00 1.200000e+01
5 3.000000e+00 2.000000e+00 1.000000e+00 1.200000e+01
1 2.000000e+00 4.000000e+00 0.000000e+00 8.000000e+00
3 3.000000e+00 3.000000e+00 0.000000e+00 5.000000e+00
7 1.000000e+00 2.000000e+00 0.000000e+00 0.000000e+00
4 3.000000e+00 5.000000e+00 1.000000e+00 0.000000e+00
6 5.000000e+00 3.000000e+00 4.000000e+00 0.000000e+00
8 2.000000e+00 4.000000e+00 9.000000e+00 0.000000e+00
```

The screen dump should show the following:

Width: 1.100000e+01
Height: 1.500000e+01
X-coordinate: 9.000000e+00
Y-coordinate: 0.000000e+00

Submission:

The project requires the submission (through Blackboard) of the C-code (source and include files), and a report explaining the time complexity of constructing a binary tree from an input file and the time complexity of computing the coordinates of the rectangles for the given binary tree. Your report should not be longer than 1 page and should be in pdf format. You should create a zip file called proj3.zip that contains all of the above and submit the zip file.

Grading:

The grade depends on the correctness of your program, clarity of your program documentation and report.

The report will account for 10% of the entire grade. The entire program will account for 90% of the entire grade. We do not test individual functions because you have the flexibility to design the structures required for this assignment.

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. Any memory leak AND/OR memory error will result in at least 50% penalty.

Getting started:

We provide two sample input files (r0_po.txt and r6_po.txt) for you to evaluate the run times of your packing program. r0_po.txt is the 3-rectangle example and r6_po.txt is the 8-rectangle example. In the 3-rectangle example, x has label 3, y has label 1, and z has label 2.

We also provide the sample output files for both examples: r0.flr and r6.flr .

We re-directed the screen output into files. The files are r0.log and r6.log .

Copy over the files from the Blackboard website. Check out the Blackboard website for any updates to these instructions.

Start packing!