ECE36800 Assignment 6

Due Tuesday, Oct 31st, 2024, 1:00pm

Extended to Thursday, Nov 7th, 2024, 1.00pm

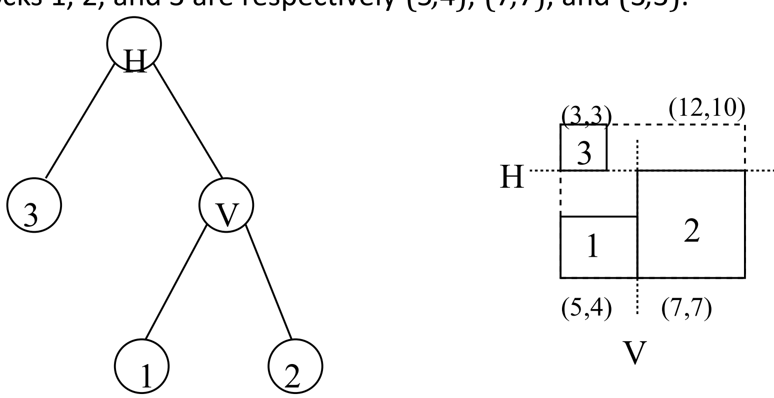
This assignment covers learning objective 1: An understanding of basic data structures, including stacks, queues, and trees; learning objective 5: An ability to design and implement appropriate data structures and algorithms for engineering applications.

You will implement a program to compute the "2D packing" of *rectangular blocks*. The "packing" of the rectangular blocks must follow a topology described using a strictly binary tree. A strictly binary tree is a binary tree where a node has either 0 or 2 child nodes. A node with 0 child nodes is a leaf node and a node with 2 child nodes is an internal (non-leaf) node.

"Packing"

In this strictly binary tree, each leaf node represents a rectangular block. Each internal node of the binary tree represents a partitioning of two groups of rectangular blocks by a horizontal cutline or a vertical cutline. Let *xHy* (*xVy*) 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 above and *y* is below the horizontal cut, and *xVy* means *x* is to the left of and *y* is to the right of the vertical cut.

In the following figure, we show a "packing" of three rectangular blocks based on a given strictly binary tree representation. Assume that the dimensions (width, height) of the three rectangular blocks 1, 2, and 3 are respectively (5,4), (7,7), and (3,3).



(a) A binary tree (b) The corresponding packing

Each subtree (whose root node is an internal node) is enclosed by a *smallest rectangular room*. The smallest room containing the subtree 1*V*2, for example, is of dimensions (12,7). The smallest room containing the tree 3*H*(1*V*2) is of dimensions (12,10). This room is partitioned into two smaller rooms: The top room is of dimensions (12,3) and it contains the rectangular block 3. The bottom room is of dimensions (12,7) and it contains the rectangular blocks 1 and 2. We place the lower left corner of each rectangular block at the lower left corner of its room.

Assume that the lower left corner of the smallest room containing all rectangular blocks is at coordinates (0,0) (*x*- and *y*-coordinates). As mentioned earlier, this room of dimensions (12,10) is partitioned into a top room of dimensions (12,3) and a bottom room of dimensions (12,7) because the root node has a horizontal cut.

Note that the smallest rectangular room containing rectangular block 3 is of dimensions (3,3), and it can be contained in the top room of dimensions (12,3). The smallest rectangular room containing the *V* node is of dimensions (12,7) and it can be contained in the bottom room of dimensions (12,7).

The bottom room, should also have its lower left corner at coordinates (0,0). As the bottom room is of height 7, the top room should have its lower left corner at coordinates (0,7).

The bottom room is partitioned into a left room of dimensions (5,7) and a right room of dimensions (7,7) because the corresponding non-leaf node has a vertical cut. Note that the smallest rectangular room containing rectangular block 1 is of dimensions (5,4), and it can be contained in the bottom-left room of dimensions (5,7). Of course, the smallest rectangular room containing rectangular block 2 is of dimensions (7,7), and it can also be contained in the bottom-right room of dimensions (7,7).

As the bottom room has its lower left corner at coordinates (0,0), the bottom-left room should have its lower left corner at coordinates (0,0). Because the bottom-left room is of width 5, the bottom-right room should have its lower left corner at coordinates (5,0).

Rectangular block 1, which is contained in the bottom-left room, should therefore have its lower left corner at coordinates (0,0). Rectangular block 2, which is contained in the bottomright room, should have its lower left corner at coordinates (5,0). Rectangular block 3, which is contained in the top room, should have its lower left corner at coordinates (0,7).

Note that even though there is space directly above block 1 to accommodate block 3, block 3 has to stay above the horizontal cutline in the “packing,” as shown in the figure. That is the reason we use “packing” instead of packing in this document. We do not really pack the rectangular blocks tightly.

For this programming assignment, you are given a strictly binary tree representation of a “packing” of rectangular blocks. You have to determine the smallest room to enclose all rectangular blocks and their coordinates, under the conditions that the cutlines are respected and that the lower left corner of a rectangular black coincides with the lower left corner of its room.

Deliverables

In this assignment, you are to develop your own include files that define the structures you want to use and declare the functions you need to manipulate the structures. You should define these functions in your source files. Your programs should be compiled with the following command: gcc -O3 -std=c99 -Wall -Wshadow -Wvla -pedantic *.c -o a6

Again, if you supply a Makefile, we will use the command ''make a6'' to generate the executable a6. The executable a6 would be invoked as follows:

./a6 in_file out_file1 out_file2 out_file3

The executable loads the strictly binary tree from in file and saves the results into three output files: out_file1, out_file2, and out_file3.

The input file in file contains the strictly binary tree and the dimensions of the rectangular blocks. The infile corresponds to a post-order traversal of the strictly binary tree. The executable should construct the corresponding strictly binary tree and output to out_file1 a pre-order traversal of the strictly binary tree. The output files out_file2 and out_file3 store the "packing" of these rectangular implementation, with out_file2 storing the dimensions of the rectangular rooms containing the rectangular blocks and out file3 storing the coordinates of the rectangular blocks.

Format of input file

argv[1]in_file contains the name of the file that stores the strictly binary tree representation of a "packing" of rectangular blocks. The file is divided into lines, and each line corresponds to a node in the strictly binary tree.

If it is a leaf node, which is a rectangular block, it has been printed with the format

"%d(%d,%d)\n",

where the first int (specified by %d) is the label of the rectangular block, followed by the dimensions (width, height) of the rectangular block, with the second int in the line being the width of the rectangular block and the third int in the line being the height. Except for the newline character, there are no white-space characters in the line. If there are *n* rectangular blocks in the "packing," the labels are from 1 through *n*.

If it is a 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 strictly binary tree. Except for the newline character, there are no other white-space characters in each line. For the example of three rectangular blocks, the input file is in 3.po as follows:

```
3(3,3)
1(5,4)
2(7,7)
V
```

H

Format of first output file

argv[2]out file1 contains the name of the file that a6 would use to store the strictly binary tree in a pre-order traversal fashion. The format of this file should be similar to that of the input file except the order in which you print the nodes. The file is divided into lines, and each line corresponds to a node in the strictly binary tree.

If it is a leaf node, which is a rectangular block, it should be printed with the format

"%d(%d,%d)\n",

where the first int is the label of the rectangular block, the second int is the width of the rectangular block and the third int is the height of the rectangular block.

If it is a 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.

Except for the newline character, there are no other white-space characters in each line. For the example of three rectangular blocks, the first output file is in 3.pr as follows:

H
3(3,3)
V
1(5,4)
2(7,7)

Format of second output file

argv[3]out file2 contains the name of the file that a6 would use to store the dimensions of all rectangular blocks (leaf nodes) and smallest rectangular rooms (non-leaf nodes or internal nodes) of a "packing." As in the input file, the nodes are printed in a post-order traversal of the strictly binary tree.

As before, if it is a leaf node, which is a rectangular block, it should be printed with the format

"%d(%d,%d)\n",

where the first int is the label of the rectangular block, the second int is the width of the rectangular block and the third int is the height of the rectangular block. If it is a non-leaf node, it should be printed with the format

"%c(%d,%d)\n",

where the char is either 'V' or 'H', representing either a vertical cutline or a horizontal cutline, respectively. The first and second int's are the width and height, respectively, of the smallest rectangular room to enclose all rectangular blocks in the subtree whose root is the non-leaf node.

Except for the newline character, there are no other white-space characters in each line. For the example of three rectangular blocks, the second output file is in 3.dim as follows:

3(3,3)
1(5,4)
2(7,7)
V(12,7)
H(12,10)

Format of third output file

argv[4]out file4 contains the name of the file that a6 would use to store the coordinates of all rectangular blocks (leaf nodes) of a “packing.”

The file should contain a line for each rectangular block. The ordering of the blocks in the output file should be the same as the ordering of blocks in the input file. Every line is of the format "%d((%d,%d)(%d,%d))\n", where the first int specifies the label of the rectangular block. The first (%d,%d) corresponds to the dimensions (width, height) of the rectangular block. The second (%d,%d) corresponds to the *x*- and *y*-coordinates of the bottom left corner of the rectangular block in the “packing.”

Except for the newline character, there are no other white-space characters in each line. For example, 3.pck stores this output as follows:

3((3,3)(0,7))
1((5,4)(0,0))
2((7,7)(5,0))

Submission

The assignment requires the submission through Gradescope.

How to make a zip file without a Makefile:

```
zip a6.zip *.c *.h
```

Your zip file should not contain a folder. You may also include a Makefile in the zip file. In that case, you can create a6.zip as follows:

```
zip a6.zip *.c *.h Makefile
```

Grading

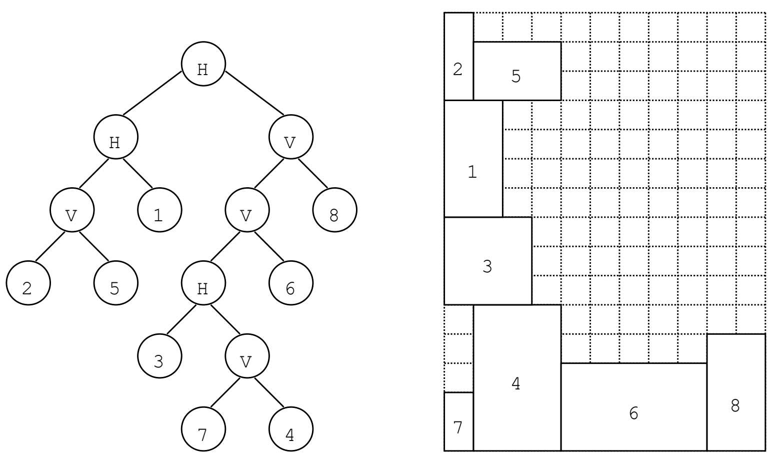
The grade depends on the correctness of your program and the efficiency of your program. The first output file accounts for 30 points, and the second output file accounts for 30 points, and the third output file accounts for 40 points of the entire grade. Any output files that do not follow the formats specified in this assignment will be considered to be wrong.

It is important that your program can accept any legitimate filenames as input or output files. Even if you cannot produce all output files correctly, you should still write the main function such that it produces as many correct output files as possible. Any output files that you cannot produce, you should leave them as empty file or not create them at all.

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 issues (memory leaks or memory errors reported by valgrind) will result in 50% penalty.

Be aware that we set a time-limit for each test case based on the size of the test case. If your program does not complete its execution before the time limit for a test case, it is deemed to have failed the test case.

What you are given



We provide two sample input files (3.po and 8.po) for you. The corresponding first, second, and third output files for 3.po are 3.pr, 3.dim, and 3.pck, and those for 8.po are 8.pr, 8.dim, 8.pck. The figure in the next page shows the topology and “packing” of the 8-block example.

Three more input files are provided (100.po, 500.po, 1K.po). However, we do not provide you the corresponding output files.