ECE 51220 Programming Assignment 2

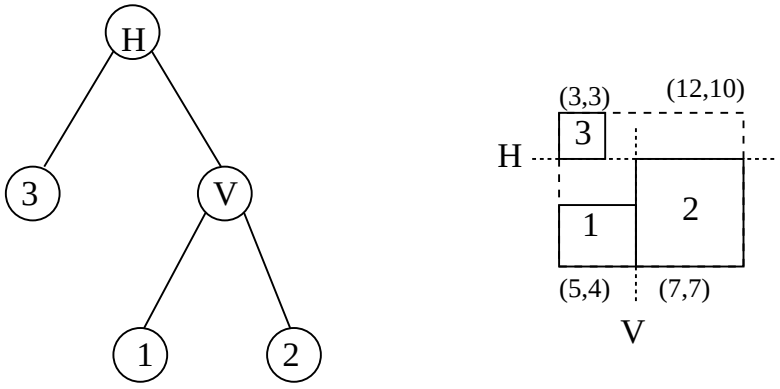
Due Monday, October 30, 2023, 11:59pm

Description

You will implement a program to compute the "2D packing" of *rectangular blocks*. Each rectangular block can have several implementations. For example, if a rectangular block has an area of 16 square units, there could be 7 possible *rectangular implementations*, with the (width, height) dimensions of the 7 implementations being $(1,16)$, $(2,8)$, $(3,6)$, $(4,4)$, $(6,3)$, $(8,2)$, $(16,1)$, where each implementation has an area ≥ 16 square units.

The "packing" of the rectangular blocks must follow a topology described using a strictly binary tree. 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 binary tree representation. We assume that each rectangular block has only one possible rectangular implementation. In particular, the dimensions (width, height) of the three respective rectangular implementations of 1, 2, and 3 are $(5,4)$, $(7,7)$, and $(3,3)$.



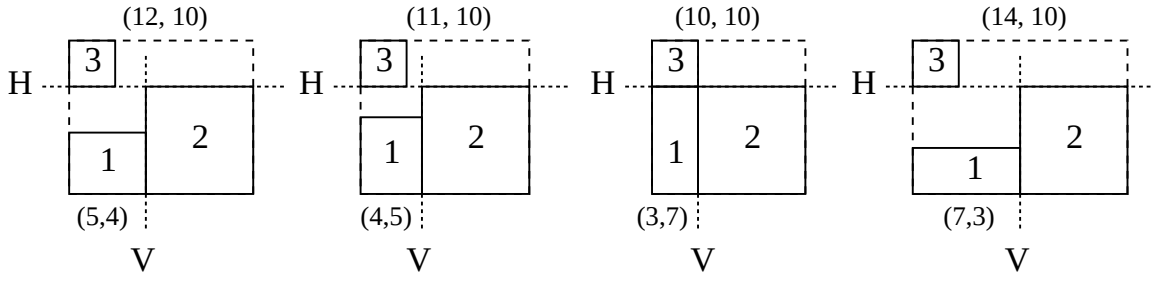
(a) A binary tree (b) The corresponding packing

Each subtree is enclosed by a *smallest rectangular room*. For example, the smallest room containing the subtree $1V2$ is of dimensions $(12,7)$. The smallest room containing the tree $3H(1V2)$ 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 implementation of block 3. The bottom room is of dimensions $(12,7)$ and it contains the rectangular implementations of blocks 1 and 2. We place the lower left corner of each rectangular implementation at the lower left corner of its room.

Assuming that the lower left corner of the smallest room containing the rectangular implementations of all blocks is at coordinates $(0,0)$ (x- and y-coordinates), the coordinates of the lower left corners of the three rectangular implementations of blocks 1, 2, and 3 are respectively $(0,0)$, $(5,0)$,

and $(0,7)$. 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" (in quotes) instead of packing in this document. We do not really pack the rectangular blocks tightly.

In the above example, each rectangular block has only one rectangular implementation. Now, consider the case that block 1 has four rectangular implementations of dimensions $(5,4)$, $(4,5)$, $(3,7)$, and $(7,3)$. In this case, there are four possible ways of "packing" the tree $3H(1V2)$. One of the four possible "packings" has a smallest rectangular room of dimensions $(12,10)$ when the rectangular implementation of block 1 is of dimensions $(5,4)$ (as shown in the preceding example). The second possible "packing" has a smallest rectangular room of dimensions $(11,10)$ when the rectangular implementation of block 1 is of dimensions $(4,5)$. In this "packing," the coordinates of the lower left corners of the three rectangular implementations of blocks 1, 2, and 3 are respectively $(0,0)$, $(4,0)$, and $(0,7)$. When the rectangular implementation of block 1 is of dimensions $(3,7)$, the "packing" has a smallest rectangular room of dimensions $(10,10)$ and the coordinates of the lower left concerns of the three rectangular implementations of x, y, and z are respectively $(0,0)$, $(3,0)$, and $(0,7)$. When the rectangular implementation of block 1 is of dimensions $(7,3)$, the "packing" has a smallest rectangular room of dimensions $(14,10)$ and the coordinates of the lower left concerns of the three rectangular implementations of blocks 1, 2, and 3 are respectively $(0,0)$, $(7,0)$, and $(0,7)$. The four "packings" are shown in the following figure.



For this programming assignment, you are given a strictly binary tree representation of a "packing" of rectangular blocks. You are also given the some rectangular implementations of each block. You have to determine the rectangular implementation of each block such that the dimensions of the smallest rectangular room containing the overall "packing" has the smallest area. For the preceding example, your program should pick, out of the four implementations for block 1, the implementation with dimensions of $(3,7)$.

Deliverables

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

```
gcc -std=c99 -pedantic -Wvla -Wall -Wshadow -O3 *.c -o pa2
```

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 or source files. The main function should reside in one (and only one) of the source files.

Again, if you supply a `Makefile`, we will use your makefile and the following command to generate the executable `pa2`:

```
make pa2
```

The executable `pa2` would be invoked as follows:

```
./pa2 in_file out_file1 out_file2 out_file3 out_file4
```

The executable loads the binary tree from `argv[1] in_file` and saves the results into four output files: `argv[2] out_file1`, `argv[3] out_file2`, `argv[4] out_file3`, and `argv[5] out_file4`. Of course, `argv[1]` through `argv[5]` should contain any valid filenames.

The input file (`argv[1] in_file`) contains the strictly binary tree and the candidate rectangular implementations of all blocks. Based on the first listed rectangular implementation of each block, the output files `argv[2] out_file1` and `argv[3] out_file2` store the "packing" of these rectangular implementation. The executable also determines the rectangular implementation for each block for the optimal "packing". The output files `argv[4] out_file3` and `argv[5] out_file4` store the optimal "packing." If there are multiple optimal "packings," any of them would be acceptable.

Format of input file

`argv[1] in_file` contains the name of the file that stores the strictly binary tree representation of "packing" of rectangular blocks. The 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 rectangular block), it has been printed with the format

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

where the `int` is the label of the rectangular block, followed by a list of dimensions of the rectangular implementations of the block, where in each `(%d,%d)`, the first `int` is the width of the rectangular implementation and the second `int` is the height. There are no spaces between each `(%d,%d)`. 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 binary tree. For the example of three rectangular blocks where block 1 has four rectangular implementations, the input file is in `3.txt` as follows:

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

Format of first output file

`argv[2] out_file1` contains the name of the file that `pa2` would use to store the dimensions of a "packing." For this "packing," `pa2` would use the first rectangular implementation of each

rectangular block in the input file. For example, the rectangular implementations of rectangular blocks 1, 2, and 3 are of dimensions (width, height) $(5,4)$, and $(7,7)$, and $(3,3)$, respectively. As described earlier, the smallest rectangular room containing the "packing" of these implementations is of dimensions $(12,10)$. You should print to the first output file using the format `"(%d,%d)\n"`, where the first `int` is the width and the second `int` is the height.

For example, 3_1.dim stores this output as follows:

```
(12,10)
```

Format of second output file

`argv[3] out_file2` contains the name of the file that `pa2` would use to store the dimensions and coordinates of rectangular implementations of a "packing." As in the case of the first output file, `pa2` would use the first rectangular implementation of each rectangular block in the input file.

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 `int` specifies the label of the rectangular block. The first `(%d,%d)` corresponds to the dimensions (width, height) of the (first) rectangular implementation of the block. The second `(%d,%d)` corresponds to the x- and y-coordinates of the bottom left corner of the rectangular implementation in the "packing."

For example, 3_1.pck stores this output as follows:

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

Formats of third and fourth output files

The format of the third output file (`argv[4] out_file3`) is the same as that of the first output file, and the format of the fourth output file (`argv[5] out_file4`) is the same as that of the second output file. The first and second output files contain information about the "packing" using the first rectangular implementation of each rectangular block. The third and fourth output files should contain information about the "packing" that has the smallest overall area among all possible combinations of rectangular implementations. For the same example `3.txt`, this corresponds to the "packing" that uses the rectangular implementation of dimensions $(3,7)$ for block 1.

The corresponding dimensions of the smallest rectangular room containing the optimal "packing" are stored in `3_all.dim`

```
(10,10)
```

The corresponding dimensions and coordinates of the rectangular implementations are stored in `3_all.pck`.

```
3((3,3)(0,7))
1((3,7)(0,0))
2((7,7)(3,0))
```

Submission

The assignment requires the submission (through Brightspace) of a zip file called `pa2.zip` that contains the source code (`.c` and `.h` files). For example, if you have only the necessary source and include files in the current directory, you can create the zip file as follows:

```
zip pa2.zip *.[ch]
```

A zip file created in this fashion does not contain a folder. You can add a `Makefile` to the zip file:

```
zip pa2.zip Makefile
```

If your `pa2.zip` contains a `Makefile`, we will use your `Makefile` to create `pa2` using the command:

```
make pa2
```

You should always copy the created zip file to a different directory, unzip the file in that directory, compile the program in that directory, and run some test cases in that directory. Submit the zip file only after you are certain that you have the correct zip file.

Your zip file should not contain a folder or directory (that contains the source files, include files, and Makefile). The zip file should contain only the source files, include files, and Makefile.

Grading

The grade depends on the correctness of your program and the efficiency of your program. The first and second output files account for 40 points, and the third and fourth output files account for 60 points of the entire grade.

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 reported by `valgrind` will result in a 50-point penalty.

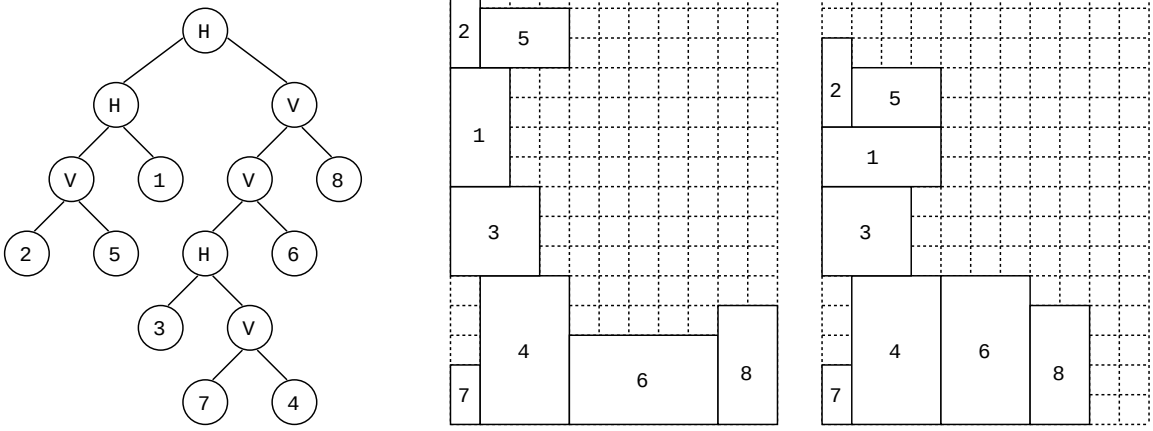
What you are given

We provide in `pa2_examples.zip` two sample input files (`3.txt` and `8.txt` in the folder/directory `examples`) for you. The corresponding first, second, third, and fourth output files for `3.txt` are `3_1.dim`, `3_1.pck`, `3_all.dim`, `3_all.pck`, and those for `8.txt` are `8_1.dim`, `8_1.pck`, `8_all.dim`, `8_all.pck`. The following shows the topology and the respective "packings" of the 8-block example.

In addition, we also provide in `pa2_examples.zip` three more sample input files without any associated output files: `10_1.txt`, `100_3.txt`, and `500_5.txt` in the folder/directory `examples`. Some of these three input files will be used for the evaluation of your submission.

Additional information

Check out the Brightspace course page for any updates to these instructions.


H
H
V
V
1
V
8
2
5
H
6
3
V
7
4
2
5
1
3
4
6
8
7
2
5
1
3
4
6
8
7