AI - 9 Nov. 2018

Block puzzles

Your homework, as discussed in class, consists of two parts. The first part, covered here, is to write a script to solve block problems, and the second is to write a script to solve rectangular Tanagrams.

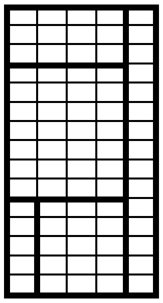
A block problem consists of several rectangles of specified height and width being assembled to give a containing rectangle. The rectangles being assembled must completely cover the containing rectangle but must not overlap. It is up to the script to determine the orientation of each rectangle.

The following example illustrates how the script should be called:

```
> block.py 15x5 3 4 4 7 15x1 1 5 5x3
Decomposition: [(3, 4), (15, 1), (7, 4), (5, 1), (5, 3)]
```

As discussed in class, the first pair of numbers represents the height and width of the containing rectangle, respectively. Every other pair of numbers represents the dimensions of one of the rectangles that should be assembled in the containing rectangle. Each pair of numbers may be separated by either a space or an `x`.

There are two situations where puzzles are unsolvable – it would be nice to differentiate between them by virtue of the text that is output. For example, [this puzzle] `Can’t be solved` or `Impossible to solve` [this puzzle] vs. [this puzzle has] `No solution`.



If the smaller rectangles do assemble into the containing one, then they should be printed out in sequence: height first, then width (see above example). The parens, brackets, and commas are optional. The way to determine the sequence is to go across each line (15 lines in the above example), and each time a new rectangle is reached, it should be output. For a given arrangement, the output sequence is unique, but block problems will normally have several solutions.

Some example puzzles that your script should be able to handle include:

```
> block.py 4 7 7 4
> block.py 2 3 1 2 2x2
> block.py 7x4 1 5 5x3 3 2
> block.py 18 9 3 11 5 7 4 8 6 10 1 2
> block.py 8x5 4x2 4 2 4x2 4 2 4x2
> block.py 55 57 28x14 32 11 32x10 21 18 21x18 21 14 21x14 17 14 28x7 28 6 10x7 14 4
> block.py 56x56 28 14 32x11 32 10 21x18 21 18 21x14 21 14 17x14 28 7 28x6 10 7 14x4
```