Computer Science AP/X — CSCI-140/242
Tripods — Lab 3

7/6/2016

1 Requirements

You will individually implement a program, `tripods.py`, that can solve any *rectangluar (N rows by M columns)* puzzle.

1.1 Command Line

The program should execute on the command line as:

```
$ python3 tripods.py filename
```

Where `filename` is the name of the text file containing the grid as one row per line, with single integers `0-9`, separated by spaces.

If the filename is not provided, or there are too many arguments, the program should display the following messsage to standard output and exit:

```
Usage: python3 tripods.py filename
```

If the filename is provided, it is guaranteed to be correctly formatted.

1.2 Program Execution

These are the steps the program should follow. Refer to the output section for the format of the messages.

1. Check the command line arguments and display/exit if the number of arguments is incorrect.
2. Read the file, save it to a collection, and display the puzzle.
3. Prompt the user for the number of optimal tripods to find. If the user asks for more tripods than can validly be placed, display an error message and exit.
4. Determine the optimal placement/s for the specified number of tripods and display them in descending order by largest sum.
5. Display the total sum of the optimal tripod placement/s.

1.3 Output Specifications

All output should happen to *standard output*.

1. If the number of command line arguments is incorrect, display the following message and exit:
   `Usage: python3 tripods.py filename`
2. Otherwise, read the file and display the puzzle (one row per line, with one space between each digit).
3. Next, prompt the user for the number of tripods to place and read the value from standard input (denoted with curly braces below):
   `Enter number of tripods: {value}`

   Here, {`value`} is guaranteed to be a valid positive integer.
4. If the requested number of tripods is too many, display the following message and exit:
   `Too many tripods to place!`
5. Otherwise, display a header message for the optimal tripod placements:
   `Optimal placement:`
6. Next, display the requested number of tripods, starting from the highest overall sum and descreasing to the next highest until the number of tripods is satisfied. *In the case of a tie, you can choose any tripod with the same overall sum.* The tripods should be displayed one per line as:
   `loc: (r,c), facing: F, sum: #`
   (a) For the tripod location, `r` is the row number and `c` is the column number. The first cell in the grid is at coordinate `(0,0)`.
   (b) For the tripod facing value, `F`, it should be a single character corresponding to the orientation:

Orientation	**Value**
North	N
East	E
South	S
West	W

   (c) The tripod sum, `#`, should be a positive integer indicating the sum of the three cells the tripod legs stand in.
7. Finally, display the sum of all the tripods, where `#`, is a positive integer:
   `Total: #`

Refer to the sample runs on the next page for examples of all the output messages.

1.4 Sample Runs

```
$ python3 tripods.py
Usage: python3 tripods filename

$ cat input-1.1
0 3 7 9
2 5 1 4
3 3 2 1
4 6 8 4
$ python3 tripods.py input-1.1
0 3 7 9
2 5 1 4
3 3 2 1
4 6 8 4
Enter number of tripods: 4
Optimal placement:
loc: (1,2), facing: N, sum: 16
loc: (3,1), facing: N, sum: 15
loc: (2,1), facing: W, sum: 14
loc: (0,2), facing: S, sum: 13
Total: 58

$ python3 tripods.py input-1.1
0 3 7 9
2 5 1 4
3 3 2 1
4 6 8 4
Enter number of tripods: 13
Too many tripods to place!

$ python3 tripods.py input-1.1
0 3 7 9
2 5 1 4
3 3 2 1
4 6 8 4
Enter number of tripods: 0
Total: 0
```

2 Implementation Details

It is strongly suggested that you follow the suggestion from the in-lab activity and use a `namedtuple` to represent a tripod.

As per the in-lab activity, you should implement the code to sort the tripods in a separate file. Modify the supplied lecture code, `rit_sort.py`, accordingly.

For full design points, you should break your program up into several functions that are driven by the main function. For example, you should have a function to read the file into a 2-D list, as well as a function to generate

3 Testing

Be sure to test your program for the following things:

1. An incorrect number of command line arguments should result in an error message.
2. The `10 x 10` puzzle from problem solving should work for any valid number of tripods.
3. A rectangular puzzle (non-square) should work fine.
4. A puzzle that is too small to place any tripods, e.g. `2 x 2`, should not allow any to be placed, but should only error if more than 0 tripods are requested.
5. A request to place more tripods than the puzzle can hold should result in an error message.

4 Grading

The grade breakdown for this lab is as follows:

- Problem Solving: 15%
- In-Lab Activity: 15%
- Functionality: 45%
- Error Handling: 10%
- Design: 5%
- Version Control: 5%
- Code Style and Documentation: 5%

5 Submission

Submit your individual implementation of this lab to `try` by the due date. You should submit the main program, the modified sorting program, and your latest Git log.

The `try` command for this lab is:

```
try csapx-grd lab3-1 tripods.py rit_sort.py log.txt
```

Recall that to verify your latest submission has been saved successfully, you can run `try` with the query option, `-q`:

```
try -q csapx-grd lab3-1
```