Programming Assignment 1: Part 2

Due: October 11, 2017

0 Instructions

Submit your work through Canvas. You should submit a tar file containing all source files and a README for running your project.

More precisely, submit on Canvas a tar file named lastname.tar (where lastname is your last name) that contains:

- All source files. You can choose any language that builds and runs on ix-dev.
- A file named README that contains your name and the exact commands for building and running your project on ix-dev. If the commands you provide don't work on ix-dev, then your project can't be graded and you won't receive credit for the assignment.

Here is an example of what to submit:

```
hampton.tar
  problem1-1.py
  problem1-2.py
  ...
  README
```

```
README
  Andrew Hampton

  Problem 1.1: python problem1-1.py input.txt
  Problem 1.2: python problem1-2.py input.txt
  ...
```

Note that Canvas might change the name of the file that you submit to something like lastname-N.tar. This is totally fine!

The grading for this part of the assignment will be roughly as follows:

Task	**Points**
Problem 2.1	10
Problem 2.2	20
Problem 2.3	20
Problem 2.3 (bonus)	5
TOTAL	50

2 Applications

Problem 2.1. Max
Write a function `stack_max` that returns the maximum value in a stack (assuming that the stack holds integers). The function should take a single argument, a stack, and should run in linear time with respect to the size of the stack.

Note: don't alter your stack class to accomplish this (for example, by adding new methods). The function `stack_max` can only use the three methods on the stack that have already been implemented.

As with **Problem 1.1** (from **Part 1**), write a driver program. The input format will be exactly the same as before, except there is one new possible instruction:

`max`: Print the maximum value in the stack. If the stack is empty, output *StackError*.

Example input file:

```
9
push 1
max
push 2
max
print
pop
pop
pop
max
```

Example output:

```
1
2
2 1
2
1
StackError
StackError
```

Problem 2.2. Brackets
Write a program to check whether a string of brackets is well formed. That is, given a string containing only the characters [ ] ( ) { } < >, we call it well formed if and only if it meets the following criteria:

- Each bracket is matched. For every open bracket (, [, {, and < there is a corresponding closing bracket.
- The substring contained within each matched pair is also well formed. For example, <[(])> is not well formed because the substring contained within the [ ] matched pair, which consists of the single character (, is not well formed.

Your program should take a single command-line argument, which will be a filename. The input file will contain strings of brackets. The first line of the input file will be an integer $0 \leq N \leq 10^4$ giving the number of strings. Following will be N lines, each containing a string of brackets having $1 \leq \text{length} \leq 10^5$.

For each given string, print *YES* if it is well formed or *NO* if it is not well formed.

All output should be to STDOUT. Each piece of output should be separated by a newline.

You should implement a solution that has linear time complexity in the string length.

Hint: use a stack.

Example input file:

```
4
()()[]<>
([<()>])
([)
([)]
```

Example output:

```
YES
YES
NO
NO
```

Problem 2.3. Restaurant Cycle
You are extremely hungry. Fortunately, you have found a group of N restuarants conveniently arranged in a circle, and you intend to eat all of the food at all of the restuarants.

However, you have to be a little bit careful. The restuarants are numbered 0 through $N - 1$ and, since the restaurants form a circle, you can only travel from restaurant i to restaurant $i + 1 \pmod{N}$. Since it takes some amount of energy to get from one restaurant to the next, it's possible that you could get stuck before completing the restaurant cycle! You want to figure out at which restaurant to begin so that you can travel to all of them.

Write a program to solve this problem. The program should take a single command-line argument, which will be a filename. The input file will contain information about the restaurants. The first line of the input file will be an integer $0 \leq N \leq 10^5$ giving the number of restaurants in the circle. Following will be N lines, each containing two integers $1 \leq E,\ D \leq 10^9$ separated by a single space. The integer E is how much energy you will receive by eating at that restaurant. The integer D is how much energy is required to travel to the next restaurant.

Wherever you choose to begin, you will start with zero energy. If at any point the energy required to travel to the next restaurant is greater than your current energy, then you cannot complete the cycle.

Output to STDOUT the restaurant number where you will start that allows you to travel to every restaurant in the circle. If there is more than one such restaurant, output the one with the smallest restaurant number. You are guaranteed that in every test case there will be at least one solution.

Note: because you are so hungry, your stomach has infinite capacity for food.

Bonus: For an extra 5 points added to your assignment total, implement a solution that runs in linear time with respect to the number of restaurants. Hint: use a queue.

Example input file:

```
4
4 2
1 11
10 3
18 6
```

Example output:

```
2
```

Explanation: There are 4 restaurants in the circle.

If we start at restaurant 0, then we get 4 energy and must spend 2 energy to get to the next restaurant. Then at restaurant 1, we get 1 energy (so we have 3 energy) but must spend 11 to get to the next restaurant. Since the distance is greater than our energy, we cannot continue.

Similarly, we can't start at restaurant 1.

If we start at restaurant 2, we get 10 energy and must spend 3 energy to travel to the next restaurant. Then at restaurant 3, we get 18 energy (so we have 25 energy) and must spend 6 to get to the next restaurant. Next is restaurant 0 (since the restaurants form a circle), where we get 4 energy (so we have 23 energy) and must spend 2 to get to the next restaurant. Finally, at restaurant 1 we get 1 energy (so we have 22 energy) and must spend 11 to get to the next restaurant. We have enough energy to complete the circle, so beginning at restaurant 2 is a valid solution.

Using similar reasoning, we see that starting at restaurant 3 also produces a valid solution.

Since we want the smallest restaurant number that gives a valid solution, we output 2.