Programming Project 2

Due: February 12, 2018

0 Instructions

Submit your work through Canvas. You should submit a tar file containing all source files and a README for running your project. Don't submit other files (e.g., input or output files).

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. The README should be able to be read on ix-dev with the `cat` utility. If the commands you provide don't work on ix-dev, then your project can't be graded and you'll be required to resubmit with a significant penalty.

Here is an example of what to submit:

```
hampton.tar
  README
  my_class1.py
  my_class2.py
  my_class3.py
  problem.py
  another_problem.py
  ...
```

```
README
  Andrew Hampton

  Problem 1: python problem1.py input
  Problem 2: python problem2.py input
  ...
```

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 the assignment will be roughly as follows:

Task	**Points**
Problem 1	**10**
pass given sample test case	3
pass small grading test case	3
pass large grading test case	4
Problem 2	**13**
pass given sample test case	3
pass small grading test case	5
pass large grading test case	5
Problem 3	**13**
pass given sample test case	3
pass small grading test case	5
pass large grading test case	5
Problem 4	**14**
pass given sample test case	4
pass small grading test case	5
pass large grading test case	5
TOTAL	**50**

Passing a test case means that the `diff` utility on ix-dev produces no output. Furthermore, passing the given sample test case requires actually completing the problem (not, for example, just hardcoding the given output).

1 Applications

Problem 1. Making a Collage
You want to construct a note by cutting out words from a magazine. First, though, you need to write a computer program that takes a list of words in a magazine and a list of the words you want to use in your note and answers whether it's possible to construct the note from the magazine.

Your program should take a single command-line argument, which will be a filename. The input file will contain exactly three lines. All words will contain only letters.

The first line is two integers x y separated by a space, with $1 \leq x, y \leq 10^6$. The first integer x denotes how many words are in the magazine. The second integer y denotes how many words are in the note.

The second line contains a space-separated list of the words in the magazine. The third line contains a space-separated list of the words in the note. The lists are case- and repetition-sensitive.

Your program should output (to STDOUT) the word *YES* if it's possible to construct the note from the magazine or *NO* if it's not possible, followed by a single newline.

The runtime complexity of your solution should be $O(x + y)$. Hint: to achieve this runtime, use a hashtable.

Example input file:

```
5 2
Hello Cat Dog World Fish
Hello World
```

Example output:

```
YES
```

Example input file:

```
5 2
Hello Cat Dog world Fish
Hello World
```

Example output:

```
NO
```

Example input file:

```
5 3
Hello Cat Dog World Fish
Hello Hello World
```

Example output:

```
NO
```

This problem is based on an example from the book Cracking the Coding Interview (McDowell).

Problem 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 3. Restaurant Cycle
You are extremely hungry. Fortunately, you have found a group of N restaurants conveniently arranged in a circle, and you intend to eat all of the food at all of the restaurants.

However, you have to be a little bit careful. The restaurants 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.

Your solution should have linear time complexity in the number of restaurants. Hint: use a queue to organize the restaurants into a cycle.

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.

2 Implementation

Problem 4. Min Heap
Implement a binary heap that maintains the min heap property. Recall that the min heap property means that smaller keys have higher priority. You should build the heap on top of a builtin array, list, or vector type, but don't use a builtin heap. For example, in Python you'll probably want to build your heap on top of the `list` data type, but you're not allowed to use the `heapq` or `Queue` modules.

Your heap data structure must implement the following methods with specified runtime:

`insert(X)`: Takes a single integer argument X and inserts the key X into the heap. *O(log n)*

`remove()`: Removes the key with highest priority and returns it. *O(log n)*

`look()`: Returns the key with highest priority. Does not alter the heap. *O(1)*

`size()`: Returns an integer, the number of keys in the heap. *O(1)*

`is_empty()`: Returns a boolean indicating whether the heap is empty. *O(1)*

`to_string()`: Returns a string, a space-separated list of the keys in the heap in order of the underlying array. *O(n)*

Write a driver program that takes a single command-line argument, which will be a filename. The input file will contain instructions for heap operations. The first line of the input file will be an integer $1 \leq N \leq 10^5$ giving the number of instructions. Following will be N lines, each containing an instruction. The possible instructions are:

`insert X`, where $-10^5 \leq X \leq 10^5$ is an integer: insert the key X into the heap. There is no output.

`remove`: Remove the key with highest priority. Output the removed element. If the heap is empty, output *HeapError*.

`print`: Output the contents of the heap separated by a single space in order of the underlying array. If the heap is empty, output *Empty*.

`size`: Output the number of keys in the heap.

`best`: Output the key with highest priority. Does not alter the heap. If the heap is empty, output *HeapError*.

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

Example input file:

```
12
print
best
insert 3
insert 2
insert 1
print
size
best
```

```
remove
remove
remove
remove
```

Example output:

```
Empty
HeapError
1 3 2
3
1
1
2
3
HeapError
```

Explanation:

Instruction	**Output**
print	Empty
best	HeapError
insert 3	NONE
insert 2	NONE
insert 1	NONE
print	1 3 2
size	3
best	1
remove	1
remove	2
remove	3
remove	HeapError