Due : October 30th, 2016

Overview

This assignment consists of two parts: implement a Stack built on top of a Singly Linked List, and use your Stack implementation to solve an algorithmic puzzle.

General Notes

- We have provided a Singly Linked List implementation for you that you should use. It uses int instead of char as data, so your PHW1 implementation will not work without modification.
- Do not change any method or class signatures. You should only edit inside of the functions. If your code changes any class or method names or signatures, you will receive an automatic 0. You should not implement any other functions besides the ones that are provided, unless explicitly allowed.
- If you are using Eclipse, be sure to remove the line declaring your code a package when you submit your code. Failure to do so could result in you receiving a 0 if the autograder fails.
- You should carefully consider all edge cases.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If you have a problem that is causing you to not be able to compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile.
- As mentioned in the overview, you have to answer whether the problem can be solved using the given data structures. But you may implement or use other standard Java data structures if you need them for your algorithm (e.g. arrays). You may not import any outside libraries.

Stack Implementation

A Stack is common, basic data structure that has many uses in computer science. Stacks have three main operations: push, pop, and peek. Push pushes data to the top of the Stack. Pop pops the data off the top and removes it from the Stack. Peek returns the data at the top of the Stack without popping it. In our implementation, we will have a bounded Stack that only allows a certain number of elements to be in the Stack at any time. Once this limit is reached, nothing can be pushed to the Stack until some elements are popped. Notice that you are not allowed to add, delete, or access data that is not at the top of the Stack.

Your goal is to implement a bounded Stack using a Singly Linked List. We have provided skeleton code for the Stack class as well as a working implementation of a Singly Linked List.

Directions

- Write your name and student ID number in the comment at the top of the Stack.java file. Please write your name in English letters, not 한글.
- Implement all of the required functions in the Stack class using the provided SinglyLinkedList class.
- You are not allowed to change the format of this class. You must only use the methods provided.
- Pay careful attention to the required return types.

Algorithmic Challenge

The second part of your assignment is to solve a puzzle using Stacks. We have provided skeleton code for a Solver class that you should use to implement your solution.

The problem is defined as follows:

- You are given 3 Stacks, A, B, and C, and a permutation of the integers from 0 to $N-1$, $1 \leq N \leq 10000$. Stack A and C both have size N, while Stack B has size $M, 1 \leq M \leq N$.
- Initially, Stack A is populated with the the integers 0 to $N-1$, in sorted order (meaning 0 is at the top of the Stack). Stacks B and C are both empty.
- Using the operations defined below, you must decide if the data in Stack A can be transferred to Stack C such that they end up in the same order as the provided permutation.
- If it can be done, your code should return "YES". If it could be done if M (the size of Stack B) was larger, but not with the current M, return "OVERFLOW". If it cannot be done with any M, return "NO".

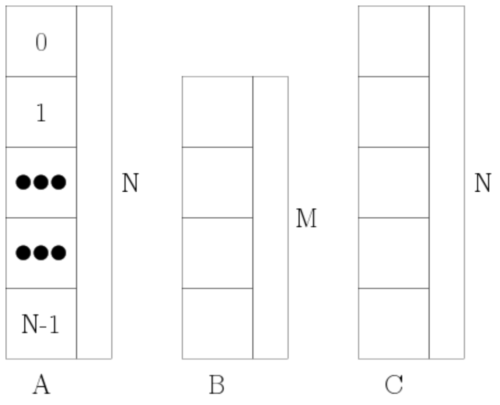
Allowed Operations

- Operation 1
  1. Pop the top element of Stack A
  2. Push the popped element to Stack B
- Operation 2
  1. Pop the top element of Stack B
  2. Push the popped element to Stack C
- Operation 1 cannot be done if Stack A is empty or Stack B is full.
- Operation 2 cannot be done if Stack B is empty.

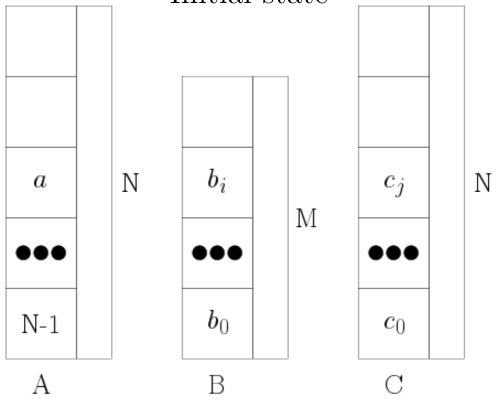
Directions

- Write your name and student ID number in the comment at the top of the Solver.java file. Please write your name in English letters, not 한글.
- Implement all of the required functions in the Solver class.
- Pay careful attention to the required return types.
- For the Solver class, you are allowed to implement your own helper functions and instance variables besides the ones that we provided. However, your solve method must still function as described in this handout.
- The data parameter in the Solver class holds the target permutation. The first int in data is what we want at the bottom of the right stack, while the last int in data is what we want at the top of the right stack. Your goal is to see if, given our stack sizes, we can move all of the elements in the left stack to the right stack (using the two allowed operations) so that they end up in the order described by the data parameter.

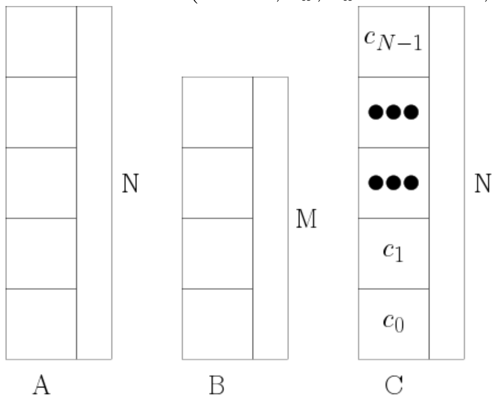
Examples



Initial state



A possible intermediate state$(0 \leq a, b_x, c_x \leq N-1, 0 \leq x \leq N-1)$



Final state$(0 \leq c_x \leq N-1, 0 \leq x \leq N-1)$

Example Inputs and Outputs

sample input	sample output
1 1	YES
0	
3 1	YES
0 1 2	
3 1	OVERFLOW
2 1 0	
3 3	YES
2 1 0	
5 5	YES
2 4 3 1 0	
5 1	OVERFLOW
2 4 3 1 0	
5 5	NO
2 0 1 4 3	
5 1	NO
2 0 1 4 3	

Deliverables

Submit **only** the following files in a zip file called PHW2.zip. Do not put them inside another directory, as this will result in you getting a 0. Be sure your code compiles and does not include the "package" line so that it will work with the autograder. This is your responsibility, not the TA's, so we will not check or correct for this error.

- Stack.java
- Solver.java

Testing

As before, we have provided a small test suite for you to check your code. The directions for how to use it can be found on the YSCEC notice board. You are provided with the following files to help you test your code.

- StackTest.java
- StackTestRunner.java
- SolverTest.java
- SolverTestRunner.java

You will need to download the junit-4.10.jar file from the YSCEC Notice Board.
You should not include any of these files in your submission.
Note that the test suite we will use to grade your code will be much more rigorous than the one provided here. You should consider making your own test cases to check your code more thoroughly.