Algorithms & Data Structures
SI3 - Polytech Nice-Sophia - Edition 2018

Lab #2: Asymtotic Analysis, ADT, Stacks, Queues

This lab will give you practice about stacks and queues.

Part 1: Big-Oh property

Suppose $T_1(N) = O(f(N))$ and $T_2(N) = O(f(N))$. Which of the following are true? Explain!

a. $T_1(N) + T_2(N) = O(f(N))$
b. $T_1(N) - T_2(N) = o(f(N))$
c. $\dfrac{T_1(N)}{T_2(N)} = O(1)$
d. $T_1(N) = O(T_2(N))$

Part 2: stack with `findMin`

In this part you have to implement another kind of stack (a stack with one more operation called `findMin`).

- Python lists can be used as optimal stacks with the method `append` and `pop`.
- Using Python lists, complete the class `StackMin` which implements a stack with the extra operation `findMin`. A stack of type `StackMin` can only handle `Comparable` object. Such a stack supports the same methods as a normal stack plus the method `findMin`: if `s` is a non-empty `StackMin`, `s.findMin()` returns the minimum value currently in the stack. The complexity of `findMin` **must** be $\Theta(1)$ (just like all the other methods). You must use the provided class template in Lab2.py
- In term of memory usage what is the worst case and when does it occur?

Supporting file:

- Lab2.py

Part 3: pairing

In this part you have first to implement a list-based queue class and then using that class, you have to solve a pairing problem.

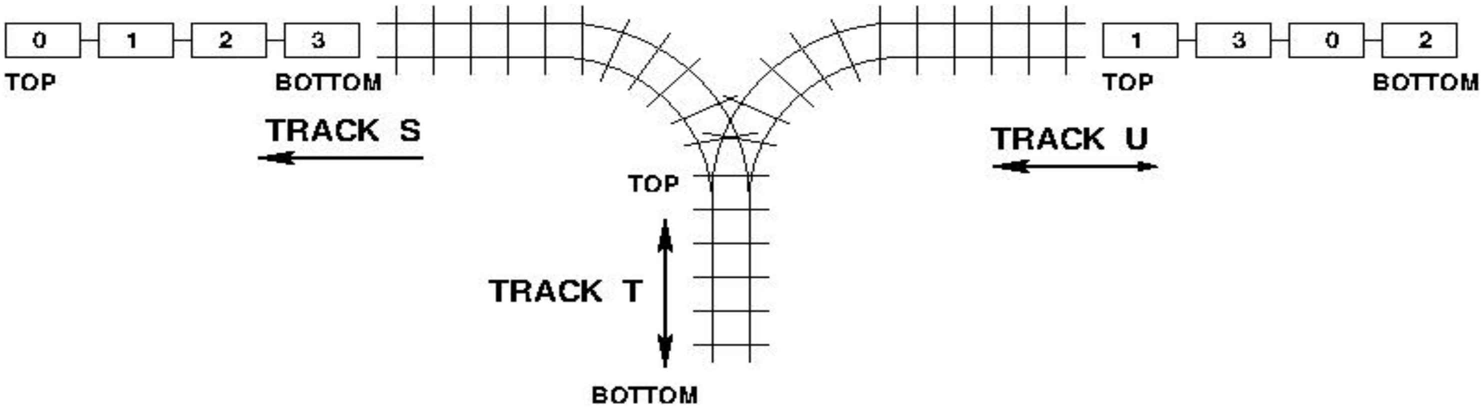
- Write the class `Queue` which implements a list-based queue, by using the provided template in `Lab2.py`.
- Using the previous class, complete the function `pairing` to solve the following problem: given an entirely increasing sequence of integers stored in a file and a constant integer N, find and display all the pairs (x,y) from the sequence such that y = x + N. For example, if N = 3 and the file contains the numbers 1 3 5 6 9 10 11 12 14 16, then the matching pairs are (3,6), (6,9), (9,12) and (11,14).
- What is the running time complexity of your algorithm?
- In term of memory usage what is the worst case and when does it occur?

Supporting file:

- big-file.txt

Part 4: train composition

You are to implement a robot to arrange train cars in a railway station. The cars are initially on track U(nsorted). Using the track T(emporary), the robot must move all the cars from track U to track S(orted) by reordering them according to a given order. The N cars making a train are always numbered from 0 to N-1.The robot can move only one car at a time from track U to track T, or from track T to track U or S. For example, in the example below, the cars are initially on track U in the order 1, 3, 0, 2 and they must be moved and reordered on track S in the order 0, 1, 2, 3:



To do so, the robot must output the following basic commands:

- move car 1 from track U to track T
- move car 3 from track U to track T
- move car 0 from track U to track T
- move car 0 from track T to track S
- move car 3 from track T to track U
- move car 1 from track T to track S
- move car 3 from track U to track T
- move car 2 from track U to track T
- move car 2 from track T to track S
- move car 3 from track T to track S

We can solve this problem using 3 stacks U, T and S. The stack U holds the cars on the track U such that the leftmost car is on top of the stack. The stack T holds the cars moving on track T. The stack S holds the cars in the final order (actually this stack is an *input* of the algorithm).

- Complete the function `train_management` such that it can return the list of basic commands needed to move and rearrange the cars in the suitable ordering.
- Ensure that both your algorithm and the sequence of basic commands produced by the arrange method are optimal (i.e. neither your algorithm or the robot are doing unnecessary move).
- Give the best and worst cases running time complexity for your algorithm and give example inputs when those cases occur.