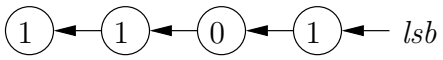
Homework Assignment #2
Due: September 20, 2024 at 5:00 p.m.

1. In class, we considered the following implementation of a counter using a linked list of bits to represent the value in binary. The pointer *lsb* points to the node containing the least significant bit and each node has a *next* pointer that points to the next bit. For example, the figure below shows the list when the counter has the value 13.



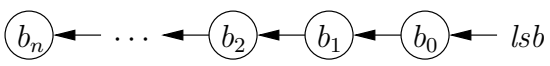

```
1: INCREMENT
2:     current ← lsb
3:     while current.value = 1 do                ▷ flip least significant bits from 1 to 0
4:         current.value ← 0
5:         if current.next = NIL then            ▷ add new node if end of list is reached
6:             current.next ← new node with value 0
7:         current ← current.next
8:     current.value ← 1                         ▷ flip rightmost 0 to 1
```

We showed that the amortized time per INCREMENT is $O(1)$. In this exercise, we consider how to add DECREMENT operations, which decrease the value stored in the counter by 1. In part (a) you will show that the representation used in class cannot do this in amortized $O(1)$ time per operation, and then in part (b) you will show that a different representation can do so.

[3] **(a)** There is an obvious way to add support for DECREMENT operations to the linked list representation: a DECREMENT simply reverses the actions of an INCREMENT. Thus, a DECREMENT flips the least significant bits from 0 to 1 until it reaches a bit that is 1. It then flips that bit to 0 and terminates. (We assume, as a precondition of DECREMENT, that the value stored in the counter is positive, which ensures that this process terminates before reaching the end of the linked list.)

Assume the counter is initialized to the value 0. For every positive integer m, give a sequence S_m of m INCREMENT and DECREMENT operations where the total number of steps taken to perform S_m is $\Omega(m \log m)$.

[6] **(b)** Now suppose that instead of storing a bit, each node of the linked list stores $-1, 0$ or 1. If the linked list contains the sequence of values $b_n, b_{n-1}, \ldots, b_0$ as shown,



then the value of the counter is $\sum_{i=0}^{n} b_i \cdot 2^i$. For example, the sequence $1, 0, -1, -1, 1$ represents the value $16 - 4 - 2 + 1 = 11$. There may be many ways to represent the same value. For example, any sequence of the form $1, -1, -1, -1, \ldots, -1$ represents the value 1.

Briefly describe how to implement INCREMENT and DECREMENT using this representation so that the amortized time per operation is $O(1)$. Then, give pseudocode for the INCREMENT

operation. Your pseudocode should ensure that there are no leading 0's in the counter, meaning if $n > 0$ then $b_n \neq 0$. Argue that both INCREMENT and DECREMENT have $O(1)$ amortized time.

[1] **(c)** Briefly discuss how using the representation described in part (b) affects the time complexity of a READ operation, which outputs a binary representation of the value stored in the register.

[3] **(d)** Briefly explain how to implement an ISZERO operation on the counter in part (b) that returns TRUE if the counter's value is 0 and FALSE otherwise. Your operation should run in $O(1)$ time in the worst case. Prove that your answer is correct.