Tutorial 3 Exercises

COMP526: Efficient Algorithms

21–22 October, 2024

Exercise 1. Recall that a Stack is an ADT that supports the functions Push, Pop, Empty, and Top. A Queue supports the methods Enqueue and Dequeue (among others). Suppose you are given two Stack instances, A and B. How could you use A and B to simulate the behavior of a Queue? That is, how can you implement Enqueue and Dequeue using *only* A and B, and the associated Stack methods for A and B?

Exercise 2. Stacks and Queues are limited in that in both cases, elements are only added to one "side" of the sequence of elements, and elements are only removed from one side. In the case of Stacks, all modifications affect only the top of the Stack. For Queues, elements are enqueued to the "back" and dequeued from the "front." We can generalize both ADTs to the Deque (pronounced "deck") ADT that allows modifications (additions and removals) to both "ends" of the sequence of elements stored in the ADT. Formally, we can represent a Deque as follows:

- The state of the Deque is a sequence S, initially $S = \varnothing$
- Append(x) modifies $S \mapsto Sx$
- AppendLeft(x) modifies $S \mapsto xS$
- Pop() modifies $Sx \mapsto S$ and returns x
- PopLeft() modifies $xS \mapsto S$ and returns x

How could you implement a Deque with an array such that all operations can be performed in $O(1)$ time? How you determine if the Deque is full? (You may assume that the size of the array is fixed so that we don't need to worry about resizing.)

Exercise 3. In Lecture 05, we described the "bubble up" procedure for adding a new element to a heap:

```
procedure Insert(p)
    v ← new vertex storing p
    u ← first vertex with < 2 children
    add v as u's child
    Parent(v) ← u
    while v not root and value(v) < value(u) do
        Swap(value(v), value(u))
        v ← u
        u ← Parent(v)
    end while
end procedure
```

Prove that the Insert procedure is correct: That is, argue that if T was a heap before calling Insert(p), then T is a heap after calling T.