# **Homework** — Algorithms and Data Structures

saved at 16:14, Friday 1st December, 2023

## Worksheet 6

1. The obvious approach would be to have the two stacks start at each end of the array $A$. For this, we can keep the indices of the tops of the stacks. When you want to push to the stack that starts at the begining of the array, you'd increment the respective stack head pointer — decrement for the other. On every push, you'd check if the stack heads have collided, which would be equivalent to stack overflow.

2. When enqueueing push the value to the frist stack. When dequeuing push every value from the first stack to the second stack while popping it fromt the first, save the topmost value of the second stack in a temporary variable, pop the second stack, go back put everything back into the first stack while emptying the second.

   The time complexity of the enqueue operation is $O(1)$ and for the dequeue $O(n)$, since it will destack and re-stack every element for every removal.

   Code in rust:

```rust
struct Queue {
    a: Vec<i32>,
    b: Vec<i32>
}

impl Queue {
    fn enqueue(&mut self, v: i32) {
        self.a.push(v);
    }

    fn dequeue(&mut self) -> Option<i32> {
        while let Some(v) = self.a.pop() {
            self.b.push(v);
        }
        let res = self.b.pop();
        while let Some(v) = self.b.pop() {
            self.a.push(v);
        }
        return res
    }
}
```

3. Linked lists can be used for this. Chaining two linked lists end to start is $O(1)$ time complexity. This will modify the internals of the first linked list.

4. If you keep the pointer to the end of the linked list, you can append a new node in $O(1)$ time, as well as proceed the head pointer to the scucessor.

5. Iterate through the linked list, setting every node's successor to be it's parent.

6. If every inner node, starting from the root, has exactly 2 children, after $d$ depth there will be $2^d$ leaves. Same with 3, if every inner node has exactly 3 children, we'll get $3^d$ leaves. These are calearly the bounds for the

amount of leaves you can get with a 2-3 tree, since no inner node can have less than 2 or greater than 3 children.

7. There are 2 possible 2-3 trees for any set consisting of 5 elements. The following are the amounts of children for every node top to bottom, left to right in a 2-3 tree: [2, 2, 3], [2, 3, 2]

   Here's one:

   Let A be the root.

$$\begin{aligned} A.s1 &= B \\ A.s2 &= C \\ A.s3 &= null \\ A.p &= null \\ A.max &= 14 \\ A.key &= null \end{aligned}$$

   the left child of the root

$$\begin{aligned} B.s1 &= B_1 \\ B.s2 &= B_2 \\ B.s3 &= B_3 \\ B.p &= A \\ B.max &= 6 \\ B.key &= null \end{aligned}$$

   the right child of the root

$$\begin{aligned} C.s1 &= C_1 \\ C.s2 &= C_2 \\ C.s3 &= null \\ C.p &= A \\ C.max &= 6 \\ C.key &= null \end{aligned}$$

   The leaves, i.e. elements of the set.

$$\begin{aligned} B_1.s1 = B_1.s2 = B_1.s3 &= null \\ B_1.p &= B \\ B_1.max = B_1.key &= 3 \end{aligned}$$

$$\begin{aligned} B_2.s1 = B_2.s2 = B_2.s3 &= null \\ B_2.p &= B \\ B_2.max = B_2.key &= 5 \end{aligned}$$

$$\begin{aligned} B_3.s1 = B_3.s2 = B_3.s3 &= null \\ B_3.p &= B \\ B_3.max = B_3.key &= 6 \end{aligned}$$

$$\begin{aligned} C_1.s1 = C_1.s2 = C_1.s3 &= null \\ C_1.p &= C \\ C_1.max = C_1.key &= 6 \end{aligned}$$

$$\begin{aligned} C_2.s1 = C_2.s2 = C_2.s3 &= null \\ C_2.p &= C \\ C_2.max = C_2.key &= 14 \end{aligned}$$