CS5800: Algorithms — — Virgil Pavlu

Homework 7cd
Submit via Gradescope

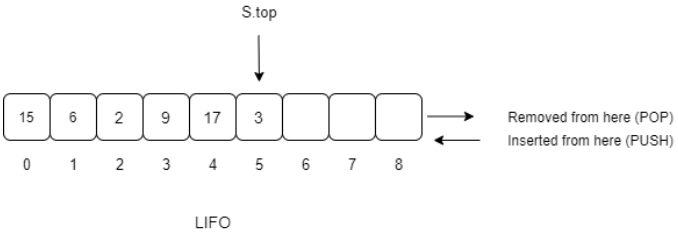
Name: Ketaki Nitin Kolhatkar
Collaborators:

Instructions:

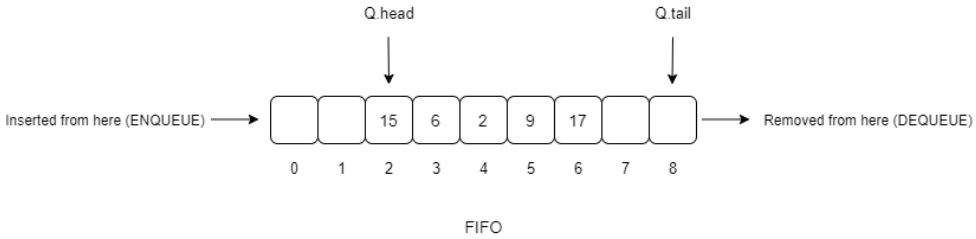
- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(30 points, Mandatory)*** *Write up a max one-page summary of all concepts and techniques in CLRS Chapter 10 (Simple Data Structures)*

In this chapter we learn about the rudimentary data structures like stacks, queues, linked lists and trees. Stacks work on a LIFO - Last In First Out policy. This means that elements that are inserted the last would be the first ones to be deleted. Elements are inserted into stacks using PUSH and deleted from the stack using POP. S.top indicates the most recently added element. We can check whether the stack is empty or not by S.top. If it is equal to 0 the stack is empty. If we try to pop an element when the stack is empty, it underflow, while when we try to push an element, it overflows.

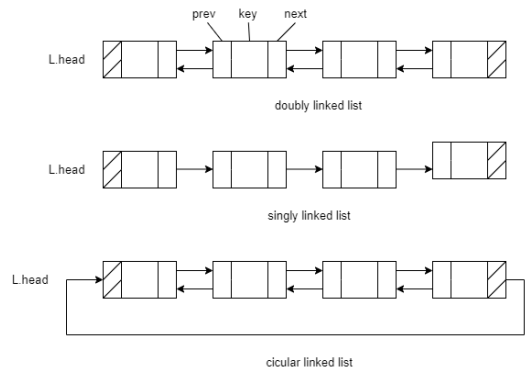


Queues work on a FIFO - First In First Out policy.
This means that elements that are inserted the last would be the first ones to be deleted. Elements are inserted into stacks using PUSH and deleted from the stack using POP.



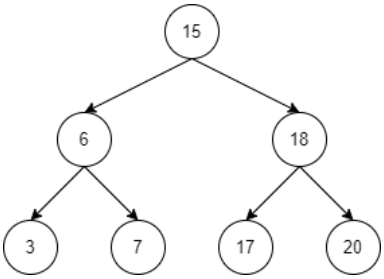
Linked list has two pointers, next and previous which point to the successor and the previous one points to its predecessor. There are various types of linked lists like singly, doubly and circular. In singly linked list one element only points to its successor element. In doubly one element points to its successor and its predecessor. In these lists, the first element's previous pointer and the last element's next pointer is empty. In circular linked list, the last element points to the first element like in a circular manner.

A binary tree has 3 pointers, parent, right child and left child. If the x.p = NIL then x is the parent, if the left subtree is absent, then x.left = NIL while if the right subtree is absent then x.right = NIL.



2. ***(30 points, Mandatory)*** *Write up a max one-page summary of all concepts and techniques in CLRS Chapter 12 (Binary Search Trees)*

Chapter 12 talks about Binary Search Trees and the operations that can be performed on them.A BST is a tree with two successors with a condition that the left child would be lesser than the root and the right child would be greater than the root. For a tree to be binary search tree, it has to satisfy this condition. There are various ways of printing all the elements of the tree like the Inorder tree walk, preorder tree walk and the post order tree walk.



Next, the chapter talks about searching - traversing through the BST. We have a pointer and keep returing values until there a key present in every node. Else if we get NIL as the answer, we stop searching through the tree.
Another operation that can be performed on the tree is finding the minimum and maximum of the elements. The minimum of the tree can be found by following the left branch of the tree until we get the key value equal to NIL. To find the maximum, we follow the right branch of the tree until we the key value equal to NIL.

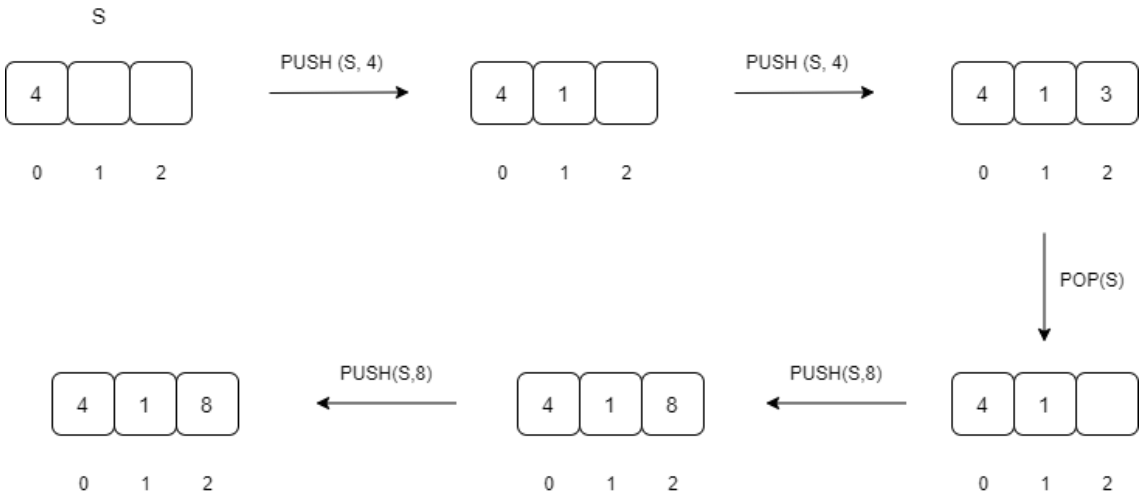
The tree successor algorithm is used to find the element which is just greater than the current element in a binary search tree. The algorithm is divided into two parts. 1.) when the element node x has right child then left most node of this right child would be the successor of the node.
2.) when the right child is empty then ancestor of the node whose left child is also ancestor would be successor. Due to similarity the overall complexity of

the method would be O(h).

For inserting a node in the tree we first find the appropriate node(parent node) i.e. this node would either be just grater or lesser than the new node. Once this place is found then we add the new node as either left or right child based upon nodes relative value with respect to parent node. The overall complexity is O(h).

For deleting a node in tree we first try to put it's children at appropriate place. If the tree just has either left or right child then that branch would either go to left or right child of node's parent. In case if the node has both child then we search the left most child of nodes right child and move all nodes to this child to it's parent and finally move the deleting node to this child. In summary we swap the deleting node with left most child of right branch and call the deletion method again. The overall time complexity of deletion is also O(h).

3. ***(10 points)*** *Exercise 10.1-1.*



4. ***(10 points)*** *Exercise 10.1-4.*

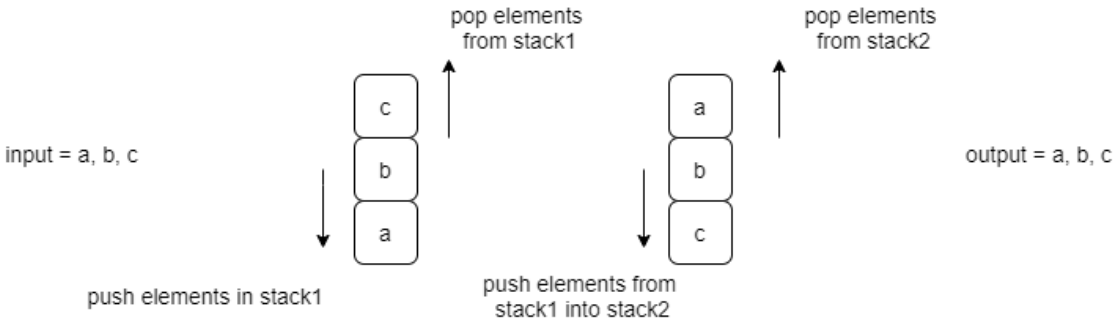
```
ENQUEUE(Q, x)
if Q.head == Q.tail+1 or Q.head == 1 or Q.tail == Q.length:
    return overflow
Q[Q.tail] = x
if Q.tail == Q.length:
    Q.tail = 1
else
    Q.tail = Q.head + 1

DEQUEUE(Q, x)
```

```
if Q.head == Q.tail+1 or Q.head ==1 or Q.tail == Q.length:
    return overflow
x = Q[Q.head]
if Q.head == Q.length:
    Q.head = 1
else:
    Q.head = Q.head + 1
return x
```

5. ***(10 points)*** *Exercise 10.1-6.*

1. We push the elements in stack 1 with a time complexity of O(1)
2. Then we pop the elements from stack 1 and push them into stack 2.
3. Next we push the elements from stack 2. This gives us a time complexity of O(n).

The output we get in the FIFO form, which is how queues work.



6. ***(Extra Credit)*** *Exercise 10.1-7.*

1. We first enqueue the elements in queue 1.
2. Then we dequeue the elements from queue 1 and enqueue them in queue 2 apart from the last element.
3. We then return the element left in queue 1.

This follows the LIFO policy that stacks use. This takes a time complexity of O(n).

7. ***(10 points)*** *Exercise 10.2-2.*

```
EMPTY(List)
if List.head == NIL:
    return True
else:
    return False

PUSH(List, a)
a.next = List.head
List.head = a

POP(List)
```

```
if List.head == NIL
    return error underflow
else:
    a = List.head
    List.head = List.head.next
    return x
```

8. ***(10 points)*** *Exercise 10.2-6.*

1. We take the elements of S1 and place them in one linked list, we do the same with the elements of S2 by placing them in another linked list.
2. For getting the union, we point S1.tail to S2.head, also could be written as S1.tail.next == S2.head. This gives us the union of the sets S1 and S2.

9. ***(10 points)*** *Exercise 10.4-2.*

```
TREE(t.root)
if t.root == NIL:
    return ""
else:
    print t.root.key
    TREE(t.root.left)
    TREE(t.root.right)
```

10. ***(Extra Credit)*** *Problem 10-1.*

	unsorted singly linked list	sorted singly linked list	unsorted doubly linked list	sorted doubly linked list
SEARCH(L, k)	O(n)	O(n)	O(n)	O(n)
INSERT(L, k)	O(1)	O(n)	O(1)	O(n)
DELETE(L, k)	O(n)	O(n)	O(1)	O(1)
SUCCESSOR(L, k)	O(n)	O(1)	O(n)	O(1)
PREDECESSOR(L, k)	O(n)	O(n)	O(n)	O(1)
MINIMUM(L)	O(n)	O(1)	O(n)	O(1)
MAXIMUM(L)	O(n)	O(n)	O(n)	O(1)

11. ***(10 points)*** *Exercise 12.2-5.*

A binary search tree has two children - left and right. When we say that the node has a left sub tree, that means that there is an element lesser than the node element and that the node element is not the minimum one. Hence it should not have a left subtree. The predecessor has to be the maximum element and so it can not have a right subtree.

12. ***(10 points)*** *Exercise 12.2-7.*

We can do this by traversing an edge twice. The number of vertices in a tree is more than the number of edges. Suppose there is a Binary search tree - x, then the edge between the x.p and x is used when the successor is called and when the largest number is called which is at x. We only use the edge twice, hence the runtime is O(n).

13. ***(10 points)*** *Exercise 12.3-3*

In the worst case scenario we are inserting the elements in the sorted order, giving us a single chain of the tree since the next number is always greater than the previous one. This will give a time complexity of $O(n^2)$. In the best case, we get the time complexity equal to $O(nlogn)$ when the tree is almost balanced and has a height of 2^h.

14. ***(Extra Credit)*** *Problem 12-3.*

15. ***(Extra Credit)*** *Problem 10-2.*

16. ***(Extra Credit)*** *Problem 15-6.*