CS5800: Algorithms — Virgil Pavlu

Homework 7

Name: MUMUKSHA PANT
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 two-page summary of all concepts and techniques in CLRS Chapter 10 (Simple Data Structures)

Solution:

CHAPTER - 10 Simple Data Structures

Dynamic sets are the sets which get manipulated by the algorithm . They can grow, shrink or change over time. Operations on dynamic sets :

1.Search (A, x)
2.Insert(A,x)
3.Delete(A,x)
4.Minimum (A)
5.Maximum (A)
6. Successor (A,x)
7. Predecessor (A,x)

A. STACKS

Stacks use LIFO (last in first out approach)
The methods in stack are :
Push() to insert data
pop() to remove data
isEmpty() to check if empty

```
Stack-Empty (S)
if top[ S ] ==0
return true;

else
return false;
```

```
Pop ( x)
IF Stack-Empty
return "underflow"

else
top[ S ] = top [S]-1
return S[top{s]+1]
```

```
Push( S, x)
top[S]= top[S]+1
S[top [ S] ] =x
```

B. Queue

Queue use FIFO Approach .
The methods are:
enqueue() : to insert into Queue
deque() : to remove from Queue

```
Enqueue(Q, x)
Q[tail [ Q ] ] =x

IF tail [Q] = lengthQ
      tail [Q ]= 1

ELSE
     tail[Q]= tail[Q] +1
```

```
Dequeue (Q)
data = Q[front]
front = front+1
```

C. Linked Lists

Unlike an array these are dynamic lists. They are of 3 main types:

1. Singly Linked List – > Key + next pointer
2. Doubly Linked List – > Key + next pointer + prev pointer
3. Circular linked list – > Similar to doubly linked list, but the prev pointer of head points to the tail.

```
searching a SLL
x= L [head]

while ( x!= NIL and key[x] != k )
      x= x.next

return x
```

```
insert a SLL
x.next = head [ L]
IF head[ L ] != NIL
   prev[head[L]]
head[L] = x
prev[x] =NIL
```

```
delete in a SLL
IF prev[x] !=NIL
  next [prev [x]  ] = next [x]
  ELSE   head [ L ] = next [x ]
IF next[x] != NIL
   prev[next[x]] =. prev[x]
```

D. Sentinels Sentinels are dummy object to simplify boundary conditions. For example, suppose that we provide with list L an object L.nil that represents NIL but has all the attributes of the other objects in the list.
We can replace the reference L to NIL in the code by using sentinels L.nil

E. Pointers and objects
(1)- **A multiple-array representation of objects:** We can represent a collection of objects that have the same attributes by using an array for each attribute. A Linked List can be represented using 3 arrays - key, next and prev

(2) **A single-array representation of objects:**The single-array representation is flexible in that it permits objects of different lengths to be stored in the same array.

F. Binary tree: we use p,left,right to store pointers to the parent, left, right child in a tree T.

if p[x] = NIL , this means it is the root of the Tree.
if root[T]=NIL , means the tree is empty
if right[x]= NIL , means no right child
if left[x]=NIL , means no left child

---

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

Solution:
CHAPTER - 12 Binary Search Tree

Search trees are data structures which support dynamic sets operations. Operations on BST take time depending on the tree height. IF tree is balanced, it takes $\theta(logn)$ time in the worst case. IF tree is linear chain , the same operations will take $\theta(n)$ time.

Keys in a BST are stored so they satisfy the Binary search tree property. A BST uses **inorder tree walk algorithm** toprint all keys in a BST in sorted manner.

Inorder Traversal : left – > root – > right
Preorder Trversal : root **before** the left or right subtree
Postorder Traversal : root **after** the left or right subtree

```
INORDER -TREE- WALK (x )
IF x !=NIL
    INORDER-TREE-WALK ( left[x ] )
     print key[x]
    INORDER-TREE-WALK ( right [x ] )
```

Querying a BST

```
TREE - search (x, k )
IF x ==NIL or k=key[x]
     return x
IF k < key[x]
   return TREE-search( left [x ] , k )
ELSE
    return  TREE-search( right [x ] , k )
```

```
TREE - min (x )

while  left [ x ]  != NIL
    x= x.left

return x
```

```
TREE - max (x )

while  right [ x ]  != NIL
    x= x.right

return x
```

Predecessor and sucessor

The predecessor P of x will be in the right subtree R1 . IF there exists a right child of the predecessor P , the child would be greater than the predecessor. This means that the P is no longer the largest element in the tree . This also means the largest key in R1 is the rightmost node in that subtree, which has no right child.

The successor S of x will be in the left subtree L2 . If Successor S has left child that child will

be less than sucessor S , and S will no longer be the minimum element in that subtree. This proves, the smallest key in left subtree is the leftmost node in that subtree, having no left child.

Insertion and Deletion
Insertion in a bst takes O(h) time where h is the height of the tree.The best and the worst case runtime will depend on the shape of the tree.

Deleting a node from a bst has 3 conditions:
1. if node has no children , we just remove it .
2. if node has only 1 child, we splice out.
3. if node has 2 child, we splice out its scucessor then replace node's key's data with successor's key and data

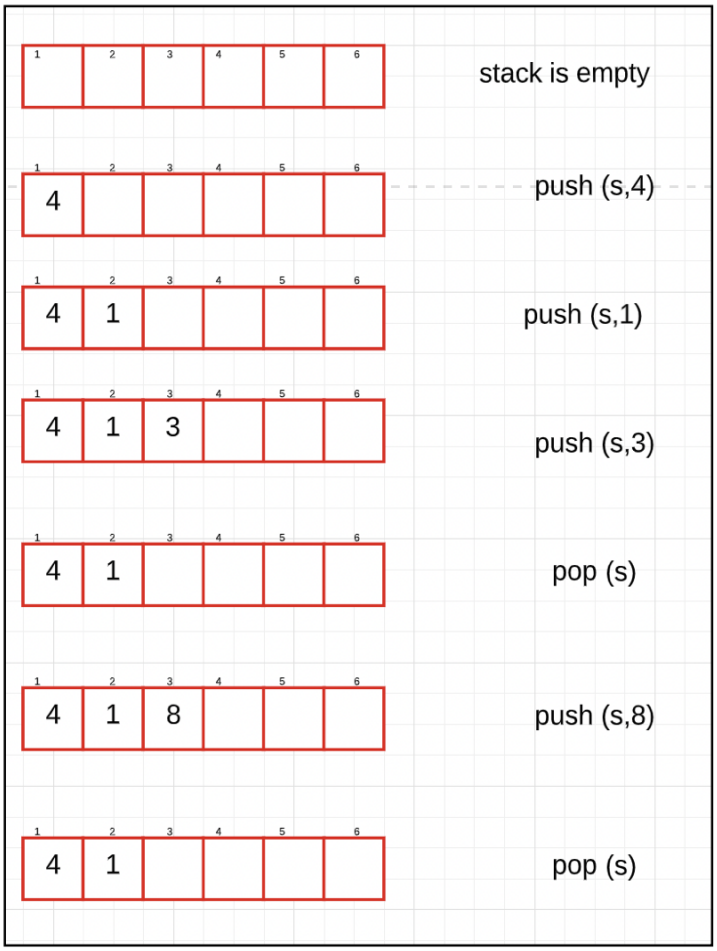
To summarise:

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

---

3. (10 points) Exercise 10.1-1.

Solution:



---

4. (10 points) Exercise 10.1-4.

Solution:

```
Q-FULL
IF Q.head==Q.tail +1
return “overflow”


Q-EMPTY
IF Q.head==Q.tail
  return “underflow” _


Procedure Enqueue (Q , x)
IF Q-FULL
   error “Overflow”

ELSE
   Q[tail [Q] ] = x

    IF Q.tail==Q.length
         Q.tail= 1

     ELSE
         Q.tail= Q.tail +1


Procedure Dequeue
IF Q.head==Q.tail
  return “underflow” _

IF Q-EMPTY
   error “underflow”

ELSE
  x= 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.

Solution:

Running time :

Push() will take O(1) time AND Pop () will take O(n) time

```
class myQueue
{
Stack S1,
Stack S2,


void Push(int data )
 { S1. push (data ) }

int pop (x)
{
   while ( ! S1. empty () )
     S2. push ( S1. pop () )

int ans = S2. pop ()

while ( ! S2.empty () )
    S1.push( S2.pop () )

return ans
}

}
```

---

6. (10 points) Exercise 10.1-7.

Solution:

Running time :

Push() will take O(n) time AND Pop () will take O(1) time

```
class myStack
{

Queue S1,
Queue S2,


void Push(int data )
 { while ( ! S1. empty () )
       S2. push ( S1. pop () )

S1.push( data)

while ( ! S2.empty () )
    S1.push( S2.pop () )
}

int pop (x)
{

return S1.pop ()
```

---

7. (10 points) Exercise 10.2-2.

Solution:

```
Push (L , x)
x.next = head.next
head=x
END

push() takes O(1) time



POP( L )
x=head.next.val
head.next=head.next.next
return x
END

pop() takes O(1) time
```

---

8. (10 points) Exercise 10.2-6.

Solution:
Take L1 and L2 as 2 circular linked lists. We will use "nil" object of the linked list

Union-Double-Linked-List (L1, L2)

```
L2.nil.next.prev= L1.nil.prev

L1.nil.prev.next = L2.nil.next

L2.nil.prev.next = L1.nil

L1.nil.prev = L2.nil.prev
```

---

9. (10 points) Exercise 10.4-2.

Solution:

Print-Tree (T)

```
x = T.root

IF  (x ! = null )
      print-tree (T.left )
      print-tree (x.key)
      print-tree (T.right)
```

---

10. (10 points) Problem 10-1.

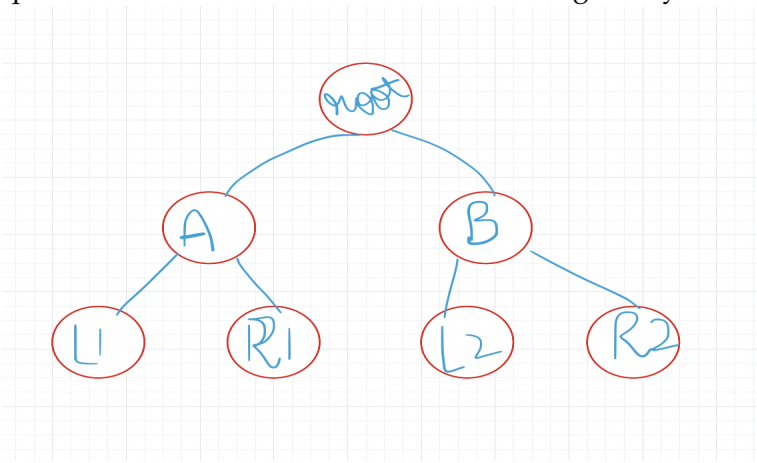
Solution:

	unsorted, singly linked	sorted, singly linked	unsorted, doubly linked	sorted, doubly linked
SEARCH(L, k)	O(n)	O(n)	O(n)	O(n)
INSERT(L, x)	O(1)	O(1)	O(1)	O(1)
DELETE(L, x)	O(n)	O(n)	O(1)	O(1)
SUCCESSOR(L, x)	O(n)	O(1)	O(n)	O(1)
PREDECESSOR(L, x)	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.

Solution:
collaborator : cs.newspaltz.edu website
"Successor" in BST is the node with the smallest key greater than key[x].
"predecessor" in BST is the node with the largest key smaller than key[x].



The predecessor P of A will be in the right subtree R1 . IF there exists a right child of the predecessor P , the child would be greater than the predecessor. This means that the P is no longer the largest element in the tree . This also means the largest key in R1 is the rightmost node in that subtree, which has no right child.

The successor S of B will be in the left subtree L2 . If Successor S has left child that child will be less than sucessor S , and S will no longer be the minimum element in that subtree. This proves, the smallest key in left subtree is the leftmost node in that subtree, having no left child.

---

12. (10 points) Exercise 12.2-7.

Solution:
1. Find the node with minimum (smallest) key, using TREE-MINMUM(x) .
2. Use TREE-Successor (x) function to find the next smallest element
3. Repeat n-1 times to iterate over entire tree.

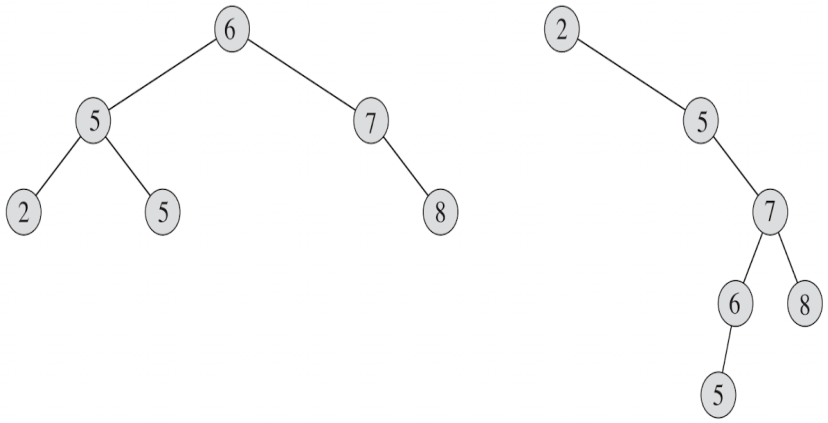
Running time of this algorithm is :
O(height of tree) + O(height of tree) X (n-1)
ignoring the smaller terms

Run time : O(height of tree * n)
height of BST tree can range from n to nlog n
This means Runtime becomes $\theta(n)$ in best case

---

13. (10 points) Exercise 12.3-3.

Solution:
The best and the worst case runtime will depend on the shape of the tree.



Best Case: $\theta(nlogn)$
When tree is balanced, that is, each node has 2 child, the height of tree will be O(logn) . Building a tree would take O(n) time AND inorder traversal will take O(log n) time.

Worst Case: $\theta(n^2)$
When tree is not balanced, and the tree is in a linear manner with inputs in decreasing order and each node have 1 child only, the tree will be less efficient. the height of tree will be O(n) .
This means building a tree will require O(n) time and inorder traversal will require $O(n^2)$

---

14. (Extra Credit) Problem 12-3.

15. (Extra Credit) Problem 10-2.

16. (Extra Credit) Problem 15-6.