Practice Problems in Data Structures

BStat, 1st Year 2023–2024

Date: 05 November, 2024

PRACTICING INSTRUCTIONS

1. Write down the best possible implementation of the problems listed below.
2. Take care of the error cases too.
3. After you finish the coding, look at the model answers at the end and understand.

NOTE: The programs are to be written in Python and should be well commented. All programs should take the required inputs from standard input buffer and print the desired outputs to the standard output buffer, until otherwise stated.

Q1. (EASY) Write an object oriented program that includes three separate classes implementing the basic functionalities of the following data structures, respectively.

(a) Stack
(b) Queue
(c) Linked List

Q2. (EASY) Given a list of numbers and an index i as user inputs, write a program to compute the nearest larger value for the number at position i. Note that `nearness` is measured in terms of the difference in array indices. For example, in the array [1, 4, 3, 2, 5, 7], the nearest larger value for 4 is 5.

Q3. (MODERATE) Let us define an *overspill stack* as a special kind of stack from which data items may automatically spill over. An overspill stack has a *spill tolerance* level of S. If S stack operations (PUSH, POP or DISPLAY) are completed since the last spill, the top element of the stack will spill over immediately **after** the completion of the S-th operation. Moreover, if the number of elements in the stack crosses its capacity, then the top element in the stack will automatically overflow because of its inherent nature. Note the following things.

(a) If the reason for a spill (i.e., the S-th operation after a spill) coincides with a natural overflow operation, only *one* element will spill over followed by the overflow.
(b) Pushing into a full stack (overflow) and popping from an empty stack (underflow) are counted as operations because they are consequences of PUSH and POP, respectively, even though they do not change the stack.

Write a program to implement the PUSH, POP and DISPLAY operations on a **generic** overspill stack, given its maximum capacity (in terms of number of elements) and spill tolerance level.

Q4. (MODERATE) Suppose the cross-product of an array A having n number of elements is defined as the following array.

$$A_{cp} = \{(A[i], A[j]) : \forall 0 < i, j < n\}$$

Given A and an integer k, find out the k^{th} lexicographically largest element in A_{cp}.

Q5. (MODERATE) Consider an array of distinct integers $A = \{a_0, a_1, \ldots, a_{n-1}\}$. One can swap any two elements of the array any number of times. An array is beautiful if the sum of $|a_i - a_{i-1}|$ among $0 < i < n$ is minimal. Given the array A, determine and return the minimum number of swaps that should be performed in order to make the array beautiful.

For example, it takes 2 swaps (between (3, 7) followed by (7, 15)) to make the array $\{7, 15, 12, 3\}$ beautiful.

Q6. (HARD) Given a binary tree, write a program to return the maximum sum of all keys of any sub-tree that is in principle a binary search tree.

Model Answers:

Q1. GENERAL APPROACH

```
class Stack():
    def __init__(self):
        self.StackDS = []
        self.SizeStackDS = 10
        print("Stack initialized")
    def StackPUSH(self, Data):
        if len(self.StackDS) < self.SizeStackDS:
            self.StackDS.append(Data)
        else:
            print("Stack overflow!!!")
        def StackPOP(self.StackDS):
        if len(self.StackDS) > 0:
            Data = self.StackDS.pop()
            return Data
        else:
            print("Stack underflow!!!")
    def StackDISPLAY(self):
        print("The elements in the stack are: ")
        for Element in self.StackDS:
            print(Element)
class Queue():
    def __init__(self):
        self.QueueDS = []
        self.SizeQueueDS = 10
        print("Queue initialized")
    def QueueINSERT(QueueDS, Data):
        if len(QueueDS) < SizeQueueDS:
            QueueDS.insert(0, Data)
        else:
            print("Queue overflow!!!")
    def QueueDELETE(QueueDS):
        if len(QueueDS) > 0:
            Data = QueueDS.pop()
            return Data
        else:
            print("Queue underflow!!!")
    def QueueDISPLAY(QueueDS):
        print("The elements in the queue are: ")
        for Element in QueueDS:
            print(Element)
class LinkedList():
    def __init__(self):
        self.LinkedListDS = []
```

```
        self.SizeLinkedListDS = 10
        print("Linked List initialized")
    def LinkedListINSERT(QueueDS, Data, Index):
        if len(LinkedListDS) < SizeLinkedListDS:
            LinkedListDS.insert(index, Data)
        else:
            print("Linked List overflow!!!")
    def LinkedListDELETE(LinkedListDS, Index):
        if len(QueueDS) > 0:
            Data = LinkedListDS.remove(Index)
            return Data
        else:
            print("Linked List underflow!!!")
    def LinkedListDISPLAY(LinkedListDS):
        print("The elements in the linked list are: ")
        for Element in LinkedListDS:
            print(Element)
```

BETTER APPROACH

```
class Stack():
    def __init__(self):
        self.StackDS = []
        self.SizeStackDS = 10
        print("Stack initialized")
    def StackPUSH(self, Data):
        if len(self.StackDS) < self.SizeStackDS:
            self.StackDS.append(Data)
        else:
            print("Stack overflow!!!")
        def StackPOP(self.StackDS):
        if len(self.StackDS) > 0:
            Data = self.StackDS.pop()
            return Data
        else:
            print("Stack underflow!!!")
    def StackDISPLAY(self):
        print("The elements in the stack are: ")
        for Element in self.StackDS:
            print(Element)
class Queue():
    def __init__(self):
        self.QueueDS = []
        self.SizeQueueDS = 10
        print("Queue initialized")
    def QueueINSERT(QueueDS, Data):
        global rear
```

```
        if (rear + 1) % SizeQueueDS == front:
            print("Queue overflow!!!")
        else:
            QueueDS[rear] = Data
            rear = (rear + 1) % SizeQueueDS
    def QueueDELETE(QueueDS):
        global front, rear, SizeQueueDS
        if front == rear:
            print("Queue underflow!!!")
        else:
            Data = QueueDS[front]
            front = (front + 1) % SizeQueueDS
            return Data
    def QueueDISPLAY(QueueDS):
        print("The elements in the queue are: ")
        if front <= rear:
            print(QueueDS[front:rear])
        else:
            print(QueueDS[front:] + QueueDS[:rear])
class LinkedList():
    def __init__(self):
        self.LinkedListDS = []
        self.SizeLinkedListDS = 10
        print("Linked List initialized")
    def LinkedListINSERT(QueueDS, Data, Index):
        if len(LinkedListDS) < SizeLinkedListDS:
            LinkedListDS.insert(index, Data)
        else:
            print("Linked List overflow!!!")
    def LinkedListDELETE(LinkedListDS, Index):
        if len(QueueDS) > 0:
            Data = LinkedListDS.remove(Index)
            return Data
        else:
            print("Linked List underflow!!!")
    def LinkedListDISPLAY(LinkedListDS):
        print("The elements in the linked list are: ")
        for Element in LinkedListDS:
            print(Element)
```