EX 8 LIST ADT USING ARRAYS

T SADAKOPA RAMAKRISHNAN | 3122225002109 | IT-B

Code:

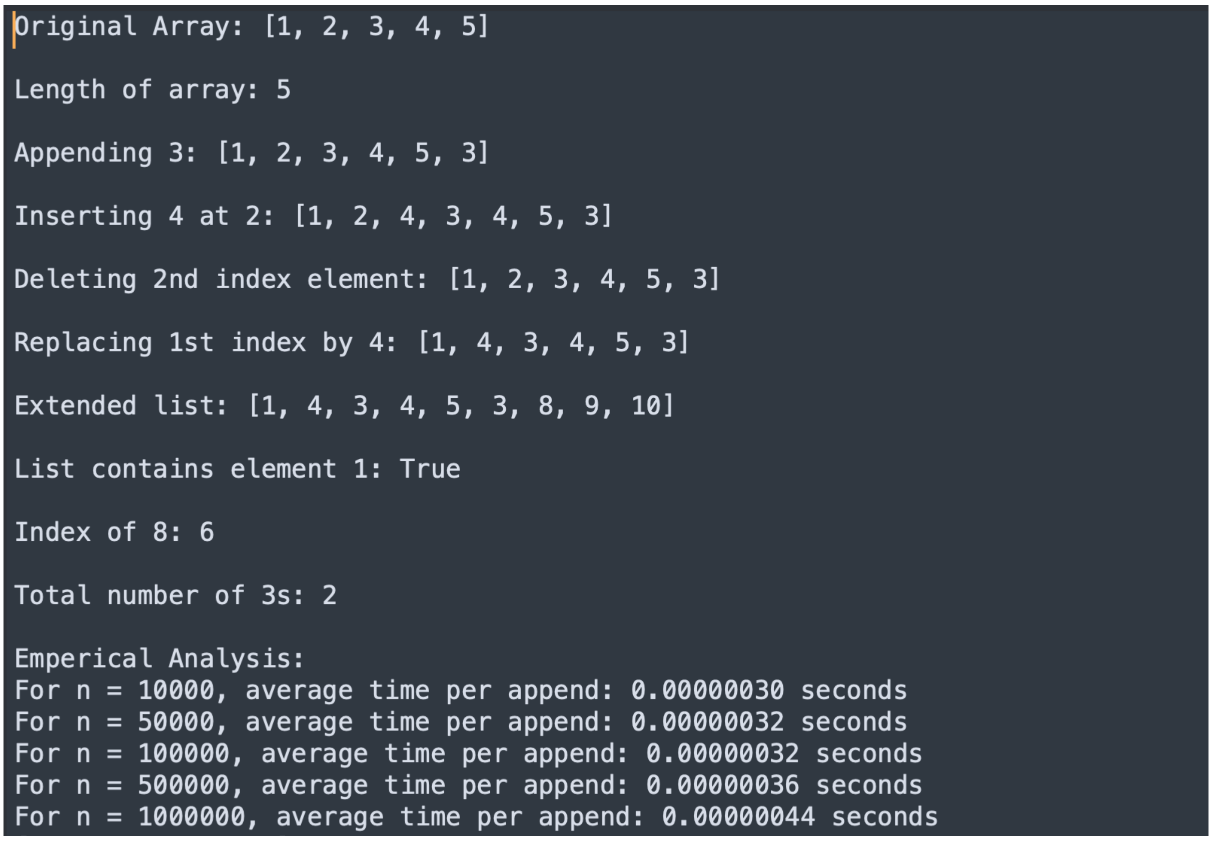
```
# -*- coding: utf-8 -*-
"""
This module provides List ADT using Array based
implementation.
Emperical analysis is done for the append method. This is
a part
of the exercises given under the course UIT2201
(Programming
and Data Structures).

In this source code I've executed my own logic and may
contain bugs.
The source code has followed good coding practices.

Your comments and suggestions are welcome.

Created on Wed May 24 2023

Revised on Wed May 29 2023

Original Author: T. Sadakopa Ramakrishnan
<sadakopa2210221@ssn.edu.in>
"""


import random
import ctypes
from timeit import default_timer as timer
class DynamicArray:
    '''
        Dynamic Array Class
    '''
    def __init__(self, val):
        '''
            Constructor to initialise variables
        '''
        if isinstance(val, int):
            self._n = 0
            self._capacity = val
            self._A = self.makearray(self._capacity)
        else:
            self._n = len(val)
            self._capacity = 2 * self._n
            self._A = self.makearray(self._capacity)
            for i in range(self._n):
                self._A[i] = val[i]

    def makearray(self, cap):
        '''
            Method to create compact arrays
        '''
        temp = (cap * ctypes.py_object)()
        return temp

    def resize(self, cap):
        '''
            Method to Resize the array
        '''
        B = self.makearray(cap)
        for i in range(self._n):
            B[i] = self._A[i]
        self._A = B
        self._capacity = cap

    def append(self, ele):
        '''
            Method to append elements to the last index
        '''
        if self._n == self._capacity:
            self.resize(2 * self._capacity)
        start = timer()
        self._A[self._n] = ele
        end = timer()
        self._n += 1
        return end - start
    def insert(self, index, ele):
        '''
            Method to insert an element at a particular
index
        '''
        if not (index <= self._n):
            raise IndexError("Index out of Range")

        if self._n == self._capacity:
            self.resize(2 * self._capacity)

        for i in range(self._n, index, -1):
            self._A[i] = self._A[i - 1]

        self._A[index] = ele
        self._n += 1

    def __str__(self):
        '''
            Method to print out DynamicArray object
        '''
        return str([self._A[i] for i in range(self._n)])

    def __len__(self):
        '''
            Method to return length of the list
        '''
        return self._n

    def __setitem__(self, index, ele):
        '''
            Method to set an element at a particular
index
        '''
        self._A[index] = ele
        return self._A

    def delete(self, index):
        '''
            Method to delete an element of a particular
index
```

```
        '''
        if not (index < self._n):
            raise IndexError("Index out of Range")

        for i in range(index, self._n - 1):
            self._A[i] = self._A[i + 1]
        self._n -= 1

        if self._n < self._capacity // 4:
            self.resize(self._capacity // 2)

    def __contains__(self,ele):
        '''
            Method to check if the list contains an
element
        '''
        for i in self._A:
            if ele == i:
                return True
        return False

    def extend(self, other_list):
        '''
            Method to append new list to the old list at
the end
        '''
        new_size = self._n + len(other_list)

        if new_size > self._capacity:
            self._resize(new_size)

        for item in other_list:
            self._A[self._n] = item
            self._n += 1

    def index(self, item):
        '''
            Method to retur a particular index of the
item
        '''
        for i in range(self._n):
            if self._A[i] == item:
                return i
        raise ValueError(f"{item} not found in the
list.")

    def count(self, item):
        '''
            Method to count a particular item
        '''
        count = 0
        for i in range(self._n):
            if self._A[i] == item:
                count += 1
        return count

def measure_append_time(n):
    '''
        To find the time it takes to append elements on
to the list
    '''
    total_time = 0
    dynamic_array = DynamicArray(n)

    for i in range(n):
        ele = random.randint(1, 100)  # Generating a
random object
        time_taken = dynamic_array.append(ele)
        total_time += time_taken

    average_time = total_time / n
    return average_time


if __name__ == "__main__":

    #Creating A Dynamic array
    l1 = DynamicArray([1, 2, 3, 4, 5])
    print("Original Array:", l1)
    print()

    #Length of array
```

```
print("Length of array:", len(l1))
print()

#Appending 3
l1.append(3)
print("Appending 3:",l1)
print()

#Inserting 4 at 2nd index
l1.insert(2,4)
print("Inserting 4 at 2:",l1)
print()

#Deleting 2nd index element
l1.delete(2)
print("Deleting 2nd index element:",l1)
print()

#Replacing 1st index by 4
l1.__setitem__(1,4)
print("Replacing 1st index by 4:", l1)
print()

#Extend the list
l1.extend([8,9,10])
print("Extended list:", l1)
print()

#Checking if list contains the element:
print("List contains element 1:", l1.__contains__(1))
print()

#Finding index of 8
print("Index of 8:",l1.index(8))
print()

#Counting number of 3s
print("Total number of 3s:", l1.count(3))
print()

#Emperical Analysis
```

```
    print("Emperical Analysis:")
    n_values = [10000, 50000, 100000, 500000, 1000000]

    for n in n_values:
        average_time = measure_append_time(n)
        print(f"For n = {n}, average time per append:
{average_time:.8f} seconds")
```

Output:



```
Original Array: [1, 2, 3, 4, 5]

Length of array: 5

Appending 3: [1, 2, 3, 4, 5, 3]

Inserting 4 at 2: [1, 2, 4, 3, 4, 5, 3]

Deleting 2nd index element: [1, 2, 3, 4, 5, 3]

Replacing 1st index by 4: [1, 4, 3, 4, 5, 3]

Extended list: [1, 4, 3, 4, 5, 3, 8, 9, 10]

List contains element 1: True

Index of 8: 6

Total number of 3s: 2

Emperical Analysis:
For n = 10000, average time per append: 0.00000030 seconds
For n = 50000, average time per append: 0.00000032 seconds
For n = 100000, average time per append: 0.00000032 seconds
For n = 500000, average time per append: 0.00000036 seconds
For n = 1000000, average time per append: 0.00000044 seconds
```