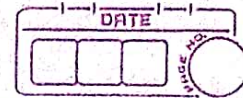


Name - Karan Kumbhar

Id No. - 12140860



* Question 1.

Class CGPA

Class Node:

slots = ("courseCode", "gradePoint", "semValue")

def __init__(self, courseCode, semValue, gradePoint):

self.courseCode = courseCode

self.gradePoint = gradePoint

self.semValue = semValue

def __init__(self, studentData, currentSemisler)

self.studentData = studentData

self.gradeToCg = {

"A+": 10,

"A": 9,

"A-": 8,

"B+": 7,

"B": 6,

"B-": 5,

"C+": 4,

"C": 3,

"C-": 2,

"D+": 1,

"D": 0,

"D-": 0

}

self.currentSemisler = currentSemisler

def SemToValue(self, data)

semDict = {

"M": 1,

"W": 2,

"S": 3

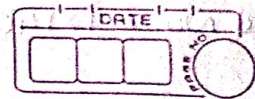
}

semValue = data.split("-")

semValue = (int(semValue[0]) + semValue[1] + int(

semDict[semValue[2]]))

return semValue



```
def makeCourseData(self):  
    requiredSemisler = self.semToValue (self._currentSemisler)  
    self._studentData.sort(key = lambda x: x[1])  
    self._courseData = []  
    prev = self._studentData[0][1]  
    for i in range (len (self._studentData)):  
        n = self.Node (self._studentData[i][1],  
            self.semValue (self._studentData[i][6]),  
            self.gradeToG [self._studentData[i][7]])  
        if n.semValue > prev:  
            if prev == self._studentData[i][1] and i != 0  
                and n.semValue <= requiredSemisler:  
                if n.semValue > self._courseData[-1].semValue:  
                    self._courseData.pop()  
                    self._courseData.append(n)  
                elif (n.semValue <= requiredSemisler):  
                    prev = n.courseCode  
                    self._courseData.append(n)  
        return self._courseData
```

```
def calculate GPA (self):  
    sum = 0  
    totalCourses = 0  
    for i in self._courseData:  
        sum += i.gradePoint  
        totalCourses += 1  
    return sum / totalCourses
```

```
if __name__ == "__main__":  
    Data = [ ['2023-24-M', 'CS102', 'F'], ['2023-24-W', 'CS102',  
        'B'] ]  
    semisler = "2023-24-W"  
    c = CGPA (Data, semisler)  
    validCourseData = c.makeCourseData()  
    Print (c.calculate GPA())
```




Here I used tree and referential array and this code taking $O(N)$ time complexity to calculate the GPA at end of any semester that you have enter

- restriction \Rightarrow semester format can't be changed
- required semester should be given
- I use this data structure as it takes minimum time.

Question 2

Class completeA:

~~slot 1~~

class Node:

slot = ("data", "left", "right", "equal",
"endWord")

def ~~init~~ (self, list):

self.n = self.Node

def ~~init~~ (self, data = None, endWord = False):

self.data = data

self.left = None

self.right = None

self.equal = None

self.endWord = endWord

def ~~init~~ (self, list)

self.n = self.Node()

self.list = list

for w in list:

self.add(w, self.n)

def add(self, w, node):

c = w[0]

if not node.data:

node.data = c

if c < node.data:

if not node.left:

node.left = ~~node~~ self.Node()

self.add(w, node.left)

elif c > node.data:

if not node.right:

node.right = ~~node~~ self.Node()

self.add(w, node.right)

else:

If len(w) == 1

node.endWord = True

return


```

if not node._equal:
    node._equal = self.Node()
    self.add(W[1:], node._equal)

```

```

def sufAll (self, pattern, node):
    if node._endWord:
        yield "{0} {1}" format (P, node._data)
    if node._left:
        for w in self.sufAll (P, node._left):
            yield w
    if node._right:
        for w in self.sufAll (P, node._right):
            yield w
    if node._leftEqual:
        for w in self.sufAll (P + node._data, node._equal):
            yield w

```

```

def finder (self, P)
    lastP = {pat: self[i] for pat in P}
    for pat in lastP.keys():
        w = self.finder_2 (pat)
        If w == None:
            return None
        else:
            comp = {Y for Y in w}
            l = list (comp)
            l1 = []
            for i in range (len (self.list)):
                if self.list[i][0] in l:
                    l1.append (self.list[i][1])
            l1.sort (reverse = True)
            l2 = []
            for i in l1:
                for j in l:
                    if (j, i) in self.list:
                        l2.append (j, i)
            return l2

```



```
def finder_2 (self, P):
    node = self._node()
    for c in P:
        while True:
            if c > node._data:
                node = node._right
            elif c < node._left._data:
                node = node._left
            else:
                node = node._equal
                break
        if not node:
            return None
    return self._offAll(OP, node)

if __name__ == "__main__":
    s = input()
    datalist = list(input())
    Pt = completeA([datalist[i][0] for i in range(len(datalist))])
    Print(Pt.finder([Cs]))
```

Here I used tree data structure, As I want to use different children (as information of parent) attached to it. This also takes minimum time.

- Time complexity = $\log(N)$
- Here datalist should be given by user

Question 3

```
def findCycleLength(self):  
    if self.head is None:  
        return 0  
    c1 = self.head  
    c2 = self.head  
    f = 0  
    while (c1 and c1.next and c2 and c2.next):  
        if c1 == c2 and f != 0:  
            count = 1  
            c1 = c1.next  
            while (c1 != c2):  
                c1 = c1.next  
                count += 1  
            return count, c1  
        c1 = c1.next  
        c2 = c2.next.next  
        f = 1  
    return 0
```

```
def act totalLength(self):  
    r, c1 = self.findlength()  
    t = self.head  
    count = 0  
    while (t.next != c1 & c1 != 'self.head'):  
        t = t.next  
        count += 1  
    return (r + count)
```