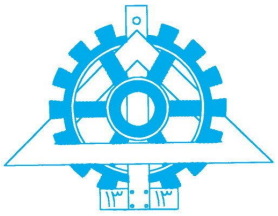
پاسخ تمرین کامپیوتری
شماره ۳



دانشکده مهندسی برق و کامپیوتر

استاد : **دکتر فتحیه فقیه**

ساختمان داده - بهار ۱۳۹۹

مسئول تمرین : غزل مینایی

مسأله ۱ : یاغی برای تمام عمر

به ازای هر الوار مساحت مستطیلی که این الوار کوتاه‌ترین عضو آن باشد را باید محاسبه کنیم. این مساحت، مساحت مستطیلی است که خانه بعد از سر و ته آن از ارتفاع فعلی کمتر باشد. برای محاسبه آن باید اولین الوار کوچکتر در چپ و اولین الوار کوچکتر در راست را به دست بیاوریم. فاصله اندیس این دو عرض و ارتفاع الوار فعلی طول مستطیل است. استکی از الوارها و اندیس‌های آن‌ها نگه می‌داریم. به ازای هر الوار جدید اگر بزرگ‌تر از سر استک است آن را در استک قرار می‌دهیم و اگر کوچک‌تر است پاپ می‌کنیم و مساحت را با کمک اختلاف اندیس‌ها و ارتفاع الوار برای آن محاسبه می‌کنیم:

۱-یک استک خالی s می‌گیریم.

۲- از اولین الوار شروع می‌کنیم، هربار:

۳-اگر s خالی است یا ارتفاع الوار فعلی بیشتر از سر استک است الوار را در استک پوش می‌کنیم.

۴-اگر کوچکتر است تا زمانی که ارتفاع بالای استک بیشتر است پاپ می‌کنیم: (تا به الوار محدودکننده برسیم)

۴-۱-با ارتفاع الوار پاپ شده مساحت را محاسبه می‌کنیم و اگر بیشتر بود مساحت قبلی را آپدیت می‌کنیم.

```
def maxAreaPoster(hists):
    stackIndex = list()
    stackHist = list()
    maxArea = 0
    index = 0
```

```
    while index < len(hists):
        currHist = hists[index]
        if (not stackIndex) or (stackHist[-1] <= hists[index]):
            stackIndex.append(index)
            stackHist.append(currHist)
            index += 1
        else:
            tsIndex = stackIndex.pop()
            tsHist = stackHist.pop()
            if stackIndex:
                area = tsHist*(index-stackIndex[-1]-1)
            else:
                area = tsHist*index
            maxArea = max(maxArea, area)

    while stackIndex:
        tsIndex = stackIndex.pop()
        tsHist = stackHist.pop()
        if stackIndex:
            area = tsHist*(index-stackIndex[-1]-1)
        else:
            area = tsHist*index
        maxArea = max(maxArea, area)
    return maxArea

n = int(input())
for i in range(n):
    inp = (input()).split()
    hist = [int(x) for x in inp]
    print(maxAreaPoster(hist))
```

مسأله ۲ : ویروس کرونا ۱۹

-لیست پیوندی را پیاده سازی می‌کنیم

-لیست پیوندی را با روش مرج سورت مرتب می‌کنیم. چون برداشتن و گذاشتن نودها در لیست پیوندی هزینه ثابت دارد مرتب سازی در $o(nlogn)$ انجام می‌شود.

-دوتا اشاره گر از اول و آخرش می‌گیریم، تا زمانی که اشاره‌گرها برابر نیستن در هر مرحله:

مجموعش اگر k شد به جواب رسیدیم

اگر کمتر شد، head را جلو می‌بریم.

اگر بیشتر شد tail را عقب می‌بریم.

از آنجا که لیست مرتب است با بردن اشاره‌گر انتهایی به چپ قطعا مقدار مجموع محاسبه شده کاهش میابد و با بردن اشاره گر ابتدایی به راست مقدار مجموع افزایش میابد. در هر مرحله اگر مجموع دو نود برابر s باشد اگر از k بیشتر باشد با جابجا کردن اشاره‌گر انتهایی مقدار آن را کاهش میدهیم. دقت کنید نودهای سمت راست اشاره‌گر انتهایی و نودهای سمت چپ اشاره گر ابتدایی را قبلا بررسی کرده‌ایم.

```
import sys
LIMIT = 1000000
sys.setrecursionlimit(LIMIT)

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

class DLL:
    def __init__(self):
        self.head = None
        self.bottom = None
    def merge(self, first, second):
        if first is None:
            return second
```

```
        if second is None:
            return first
        if first.data < second.data:
            first.next = self.merge(first.next, second)
            first.next.prev = first
            first.prev = None
            return first
        else:
            second.next = self.merge(first, second.next)
            second.next.prev = second
            second.prev = None
            return second
    def mergeSort(self, temp):
        if temp is None:
            return temp
        if temp.next is None:
            return temp

        second = self.split(temp)

        temp = self.mergeSort(temp)
        second = self.mergeSort(second)
        return self.merge(temp, second)

    def insert(self, data):
        new_node = Node(data)
        self.head = new_node

    def insertEnd(self, data):
        if self.head.next is None:
            new_node = Node(data)
            self.head.next = new_node
            self.bottom = new_node
            return

        n = self.bottom
        new_node = Node(data)
        n.next = new_node
```

```
        new_node.prev = n
        self.bottom = new_node

    def split(self, temp):
        fast = slow = temp
        while(True):
            if not fast.next:
                break
            if not fast.next.next:
                break
            fast = fast.next.next
            slow = slow.next

        temp = slow.next
        slow.next = None
        return temp

    def printList(self, node):
        temp = node

        while(node is not None):
            print (node.data)
            temp = node
            node = node.next

def findK(linkedList, k, size):
    head = linkedList.head
    tail = None
    i = 0
    while i != (size - 1):
        if i == 0:
            tail = head.next
        else:
            tail = tail.next
        i += 1

    while head and tail and head != tail:
        if (head.data + tail.data) == k:
```

```
            if head.data < tail.data:
                print(head.data, tail.data)
            else:
                print(tail.data, head.data)
            return
        else:
            if (head.data + tail.data) < k:
                head = head.next
            else:
                tail = tail.prev

    print("NO")


size, k = map(int, input().split());
inp = [int(i) for i in input().split("-")]

people = DLL()
people.insert(inp[0])
for i in range(1, len(inp)):
    people.insertEnd(inp[i])

people.head = people.mergeSort(people.head)

findK(people, k, size)
```

مسأله ۳ : ریک در قرنطینه

باید همیشه جای فعلی مکان‌نما را بدانیم برای این کار فرض می‌کنیم مکان نما سر یک استک است. و بقیه حروف نیز در یک استک دیگر قرار گرفته‌اند که سر آن دقیقا حرف بعد از مکان‌نما است. پس یک استک برای چپی‌ها (کاراکترهای سمت چپ مکان‌نما) در نظر میگیریم و یکی برای راستی‌ها (آن‌هایی که سمت راست مکان نما هستند). اگر دستور L آمد یعنی مکان‌نما باید یکی به سمت چپ برود پس از سر استک چپی‌ها یک کاراکتر برمی‌داریم و به سر استک راستی‌ها اضافه میکنیم.

برای دستور R نیز به طور مشابه، از استک راستی‌ها برمی‌داریم و به استک چپی‌ها اضافه می‌کنیم. با این کار مکان نما همیشه سر استک چپی‌ها باقی می‌ماند:

-دوتا استک می‌گیریم s1 و s2

-رشته رو از چپ پیمایش می‌کنیم.

-اگر کاراکتر R بود و s2 خالی نبود از سر s2 پاپ می‌کنیم در s1 پوش می‌کنیم.

-اگر کاراکتر L بود و s1 خالی نبود از سر s1 پاپ می‌کنیم در s2 پوش می‌کنیم.

-اگر این دو حالت نبود، کاراکتر را در s1 می‌گذاریم.

- در آخر یکی یکی از s2 پاپ می‌کنیم و در s1 پوش می‌کنیم.

```
inp = input()
s1 = list()
s2 = list()

for i in inp:
   if i == 'R' and s2:
       s1.append(s2.pop())
   elif i == 'L' and s1:
       s2.append(s1.pop())
   elif i != 'R' and i != 'L':
       s1.append(i)

while s1:
   s2.append(s1.pop())

s = ""
```

```
while s2:
    s += str(s2.pop())

print(s)
```

مسأله ۴ : عبارات دردسرساز(امتیازی)

برای تبدیل پیش‌وندی به پس‌وندی:

وروردی را از انتها پیمایش می‌کنیم. اگر کاراکتر یک عدد بود درون استک قرار می‌دهیم و در غیر این صورت دوتا داده از استک خارج می‌کنیم (x و y) و به صورت x+y+char در استک قرار می‌دهیم.

برای تبدیل میان‌وندی به پیش‌وندی:

۱-دوتا استک یکی برای اپراتورها و یکی برای اعداد درست می‌کنیم

۲-یکی یکی کاراکترهای ورودی را بررسی می‌کنیم

۲-۱-اگر عدد بود در استک اعداد قرار می‌دهیم

۲-۲-اگر) بود در استک اپراتورها قرار می‌دهیم

۲-۳-اگر (بود تا زمانی که سر است اپراتورها) نشده:

۲-۳-۱-از استک اعداد دوتا داده خارج می‌کنیم (x و y) و با یک داده از اپراتورها ترکیب می‌کنیم و در استک اعداد قرار می‌دهیم:

operator + y + x

۲-۴-در غیر این صورت:

۲-۴-۱-تا زمانی که اولویت اپرند ورودی از سر استک کمتر است

۲-۴-۱-۱-از استک اعداد دوتا داده خارج می‌کنیم (x و y) و با یک داده از اپراتورها ترکیب می‌کنیم و در استک اعداد قرار می‌دهیم:

operator + y + x

۲-۴-۲-در آخر کاراکتر ورودی را در استک اپراتورها قرار می‌دهیم

۳-در پایان تا زمانی که در استک اپراتورها چیزی باقی مانده:

۳-۱-از استک اعداد دوتا داده خارج می‌کنیم (x و y) و با یک داده از اپراتورها ترکیب می‌کنیم و در استک اعداد قرار می‌دهیم:

operator + y + x

```
def getPriority(c):
    if c == '-' or c == '+':
        return(1)
    elif c == '*' or c == '/':
        return(2)
    return(0)


def inToPre(s):
```

```
    op = list()
    n = list()
    for i in range(len(s)):
        if s[i] == '(':
            op.append(s[i])
        elif s[i] == ')':
            while len(op) and op[len(op) - 1] != '(':
                x = n.pop()
                y = n.pop()
                z = op.pop()
                result = z + y + x
                n.append(result)
            op.pop()
        elif s[i].isdigit():
            n.append(s[i])
        else:
            while len(op) > 0 and getPriority(s[i]) <= getPriority(op[len(op) -
1]):
                x = n.pop()
                y = n.pop()
                z = op.pop()
                result = z + y + x
                n.append(result)
            op.append(s[i])
    while len(op) > 0:
        x = n.pop()
        y = n.pop()
        z = op.pop()
        result = z + y + x
        n.append(result)
    return(n[len(n) - 1])

def preToPost(s):
    result = list()
    for i in range(len(s) - 1, -1, -1):
        if s[i].isdigit():
            result.append(s[i])
        else:
```

```
            x = result.pop()
            y = result.pop()
            tmp = x + y + s[i]
            result.append(tmp)
    return(result.pop())

x = input()
if getPriority(x[0]) > 0:
    print(preToPost(x))
else:
    print(inToPre(x))
```