Задача A. Постфиксная запись

Разбор

Будем поддерживать стек и во время прохода по списку с элементами выражения рассмотрим два случая:

1. Текущий элемент выражения – число, тогда поместим его на вершину стека
2. Текущий элемент выражения – операция, тогда два предыдущих элемента в стеке должны быть числами, снимем их со стека и применим к ним операцию

Если обрабатываемое выражение является корректным, то в конце в стеке останется только один элемент, который и будет являться ответом.

Асимптотика решения: $O(N)$, где N - число элементов выражения

Решение

```
vals = input().split()
ops = {
    '+': lambda x,y: x + y,
    '-': lambda x,y: x - y,
    '*': lambda x,y: x * y,
}
res = []
for v in vals:
    if v in ops:
        b, a = res.pop(), res.pop()
        op = ops[v]
        res.append(op(a, b))
    else:
        res.append(int(v))
print(res[-1])
```

Задача B. Минимум на отрезке

Напишем дек с поддержкой минимума, в котором будем хранить индексы элементов массива, входящих в текущее «окно». Пусть «окно» сдвинулось. Удаляем из дека все индексы, элементы по которым больше в «окно» не попадают. Затем удалим с конца все индексы, значения по которым меньше, чем новоприбывшее значение, и добавим новый индекс. Таким образом, в начале дека хранится индекс элемента массива, дающий минимум на текущем «окне». Если «окно» полностью вмещается в массив, выводим ответ.

Решение

```
    queue = deque()
    for i in range(N):
        while len(queue) > 0 and queue[0] < i - K + 1:
            queue.popleft()
        while len(queue) > 0 and array[queue[-1]] > array[i]:
            queue.pop()
        queue.append(i)

        if i >= K - 1:
            print(array[queue[0]])
```

Задача C. Сортировка вагонов

Заведём стек standstill - это тупик, в который будем помещать вагоны. Также понадобится счётчик number - текущий номер вагона, который ещё не перевели на путь 2. Кладём из пути 1 вагон в стек. Если его номер совпадает с number, то переводим на путь 2 и достаём из стека. При этом, если в стек пришёл вагон с номером большим, чем вершина стека, то выстроить вагоны по порядку невозможно (когда будем доставать их из стека, вагон с большим номером встанет раньше). Всё время поддерживаем массив operations, куда кладём пары (тип опнрации, номер вагона). Когда уже перевели все вагоны на путь2, просматриваем opeartions и подсчитываем, сколько было подряд идущих операций одного типа.

Решение

```
standstill = []
operations = []
number = 1

for wagon in wagons:
    if standstill and wagon > standstill[-1]:
        print(0)
        quit(0)
    operations.append(1)
    standstill.append(wagon)
    while standstill and standstill[-1] == number:
        operations.append(2)
        standstill.pop()
        number += 1
operation = operations.pop(0)
count = 1
ans = []
while operations:
    if operations[0] == operation:
        count += 1
    else:
        ans.append([operation, count])
        operation = operations[0]
        count = 1
    operations.pop(0)
ans.append([operation, count])
print(len(ans))
for i in ans:
    print(*i)
```

Задача D. Гоблины и очереди

Разбор

Для начала научимся обрабатывать запросы '+' и '-', для этого нам подойдёт обычная очередь, используем для этого класс deque из модуля collections. Теперь нам надо научиться быстро добавлять гоблина в середину.

Для этого воспользуемся идеей организации очереди на двух стеках – просто разделим очередь на две половины и будем при добавлении гоблина в середину или удалении гоблина из очереди восстанавливать равенство числа гоблинов в двух частях.

Асимптотика решения: $O(1)$ на запрос

Решение

```
from collections import deque

class Queue:
    def __init__(self):
        self.l = deque()
        self.r = deque()

    def push(self, v: int):
        self.l.appendleft(v)

    def push_mid(self, v: int):
        while len(self.l) > len(self.r):
            self.r.appendleft(self.l.pop())
        self.l.append(v)

    def pop(self) -> int:
        while len(self.l) > len(self.r):
            self.r.appendleft(self.l.pop())
        return self.r.pop()

q = Queue()
n = int(input())
for _ in range(n):
    cmd = input().split()
    if cmd[0] == '+':
        q.push(int(cmd[1]))
    if cmd[0] == '*':
        q.push_mid(int(cmd[1]))
    if cmd[0] == '-':
        print(q.pop())
```

Задача E. Хорошие дни

Найдём для каждой позиции в массиве самый длинный отрезок, содержащий этот элемент и на котором этот элемент является минимальным. Заметим, что левая и правая границы такого отрезка – это ближайшие слева и справа элементы, которые его меньше. Найти эти самые ближайшие для каждой позиции можно за два прохода со стеком. Ну а чтобы для каждого найденного отрезка за $O(1)$ находить сумму достаточно посчитать префиксные суммы.

Решение

```
def max_sum_min_subarray(arr):
    n = len(arr)
    prev_smaller = [-1] * n
    stack = []
    for i in range(n):
        while stack and arr[stack[-1]] >= arr[i]:
            stack.pop()
        if stack:
            prev_smaller[i] = stack[-1]
        stack.append(i)
```

```
left_boundaries = [-1] * n
stack = []
for i in range(n):
    while stack and arr[stack[-1]] > arr[i]:
        stack.pop()
    if stack:
        left_boundaries[i] = stack[-1]
    stack.append(i)

right_boundaries = [-1] * n
stack = []
for i in range(n - 1, -1, -1):
    while stack and arr[stack[-1]] >= arr[i]:
        stack.pop()
    if stack:
        right_boundaries[i] = stack[-1]
    stack.append(i)

prefix_sums = [0] * (n + 1)
for i in range(1, n + 1):
    prefix_sums[i] = prefix_sums[i-1] + arr[i-1]

ans = 0
for i in range(n):
    l = left_boundaries[i] + 1
    r = right_boundaries[i] - 1 if right_boundaries[i] != -1 else n - 1
    ans = max(ans, arr[i] * (prefix_sums[r+1] - prefix_sums[l] if l <= r else 0))

return ans
```