Линейные структуры данных и квадратичные сортировки

1 A. Анаграммы

Разбор

Для начала поймём, что две строки являются анаграммами, если они состоят из одинаковых символов, при этом повторяющиеся символы должны иметь одинаковое число вхождений в строку. Мы можем проверить это, отсортировав символы в обеих строках и сравнив полученные отсортированные списки. Если эти списки равны, то строки являются анаграммами.

Асимптотика решения: $O(|S| \times \log |S| + |T| \times \log |T|)$

Решение

```
s = sorted(list(input()))
t = sorted(list(input()))
print('YES' if s == t else 'NO')
```

2 B. Простая очередь

Разбор

Решим эту задачу с использованием двух стеков.

Давайте возьмем два стека b и f и будем класть все добавляемые элементы на вершину стека b. Тогда при вызове команд *front* и *pop* возможны два случая:

1. Стек f пустой, тогда нам нужно переложить все элементы из стека b в него в обратном порядке
2. Стек f непустой и тогда мы просто берём нужный элемент с его вершины

Длина очереди определяется как сумма длин обоих стеков, очистка очереди также подразумевает очистку обоих стеков.

Асимптотика решения: $O(1)$ на запрос

Решение

```
b, f = [], []
c = input()
while c != 'exit':
    cmd = c.split()
    if cmd[0] == 'push':
        b.append(cmd[-1])
        print('ok')
    if cmd[0] == 'pop':
        if len(f) == 0:
            f = list(reversed(b))
            b = []
        print(f[-1])
        f.pop()
    if cmd[0] == 'front':
        if len(f) == 0:
            f = list(reversed(b))
            b = []
        print(f[-1])
```

```
    if cmd[0] == 'size':
        print(len(f) + len(b))
    if cmd[0] == 'clear':
        f = []
        b = []
        print('ok')
    c = input()
print('bye')
```

3 C. 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])
```

4 D. Кошмар в замке

Разбор

Давайте сначала придумаем интуицию решения. Сразу выделим несколько идей:

- У каждой буквы есть свой вес – значит, надо их как-то отсортировать
- На вес всей строки также влияет расстояние между одинаковыми буквами – значит, их надо разместить в разных концах строки

Здесь нужно обратить внимание на ключевую фразу *"для каждой буквы алфавита посчитай **максимальное** расстояние между позициями"*. Так как для каждой буквы важно **максимальное** расстояние, будем размещать только одну пару каждой буквы в порядке убывания веса букв, а затем добавим между парами все остальные буквы в произвольном порядке, т.к. они уже не влияют на вес.

Асимптотика решения: $O(N \times \log N)$

Решение

```
def itoc(i):
    return chr(ord('a') + i)

s = input()
w = list(map(int, input().split()))

n = len(s)
cnt = [0] * 26
for c in s:
    cnt[ord(c) - ord('a')] += 1

chars = list(range(26))
chars.sort(key=lambda x: w[x], reverse=True)

k = 0
l, r = [], []
for c in chars:
    if cnt[c] > 1:
        l.append(itoc(c))
        r.append(itoc(c))
        cnt[c] -= 2
for c in range(26):
    while cnt[c] != 0:
        l.append(itoc(c))
        cnt[c] -= 1
r.reverse()
print(''.join(l + r))
```

5 E. 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) -> None:
        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())
```

6 F. Число

Разбор

В этой задаче важно было понять, как правильно сортировать список с "кусками" числа. Первое, что приходит в голову – сделать сортировку в лексикографическом порядке и просто склеить части. Однако тут возникает проблема: к примеру, части числа 4 и 4321 были бы отсортированы в неправильном порядке, т.к. лексикографически 4321 больше, чем 4

Можно было бы долго пытаться придумать правило вычисления ключа для сортировки с учетом реализации на Python, а можно было воспользоваться компараторами аналогично другим языкам. Поймём, какой компаратор мы бы хотели использовать. При заданных x, y мы хотим поставить x вперёд, если число xy больше, чем число yx.

Асимптотика решения: $O(N \times \log N)$

Решение

```
from functools import cmp_to_key
from sys import stdin

def cmp(x, y):
    return 1 if x + y > y + x else -1

v = []
for line in stdin:
    v.append(line.rstrip())
v.sort(key=cmp_to_key(cmp), reverse=True)
print(''.join(v))
```