Разбор задач

Задача 1. Путешествие поездом

Легко видеть, что номера мест в плацкартном купе можно найти с помощью следующих формул:

1. $(номер\_вагона - 1) * 4 + 1$;
2. $(номер\_вагона - 1) * 4 + 2$;
3. $(номер\_вагона - 1) * 4 + 3$;
4. $(номер\_вагона - 1) * 4 + 4$.

Схожим образом можно построить формулы и для вычисления номеров боковых мест. Для этого достаточно заметить, что номера боковых мест идут по убыванию, начиная с номера равного количеству мест в вагоне. Итоговые формулы для боковых мест:

5. $11 * 6 - (номер\_вагона - 1) * 2 - 1$;
6. $11 * 6 - (номер\_вагона - 1) * 2$.

Также возможно альтернативное решение — так как количество возможных тестов всего 11, то можно было в программе сохранить ответы для всех возможных тестов.

```
carriage = int(input())
print((carriage - 1) * 4 + 1)
print((carriage - 1) * 4 + 2)
print((carriage - 1) * 4 + 3)
print((carriage - 1) * 4 + 4)
print(11 * 6 - (carriage - 1) * 2 - 1)
print(11 * 6 - (carriage - 1) * 2)
```

Задача 2. Пирожки

Каждые три дня Петя будет тратить $a + b + c$ рублей вне зависимости от того, какой именно пирожок он купил первым. Поэтому можно сначала посчитать количество полных групп длительностью три дня, разделив нацело n на 3 (в Python $n//3$, в C++, C#, Java $n/3$, в Pascal n **div** 3) и вычислить сумму, потраченную за это время.

Остаток от деления целого числа (n) на 3 может принимать значения 0, 1 или 2 — это количество дней, не учтённых при вычислении суммы выше.

Таким образом, если остаток от деления составляет 0, то сумма уже известна.

Если остаток от деления равен 1, то к уже вычисленной сумме нужно добавить цену пирожка, который Петя купил в первый день.

Если же остаток от деления равен 2, то к уже вычисленной сумме нужно добавить цену пирожка, который Петя купил в первый день, и цену пирожка, который Петя купил во второй день.

Приведём одно из возможных решений (на Python):

```
a = int(input())
b = int(input())
c = int(input())
n = int(input())
start = int(input())

ans = n // 3 * (a + b + c)
if start == 2:
```

```
    a, b, c = b, c, a
elif start == 3:
    a, b, c = c, a, b

if n % 3 >= 1:
    ans += a
if n % 3 >= 2:
    ans += b
print(ans)
```

Решения, не использующие формулу для подсчёта суммы, потраченной за количество дней, кратное 3, а пошагово симулирующие процесс, не укладываются в ограничения по времени и набирают не более 72 баллов. Пример такого решения (на Python):

```
a = int(input())
b = int(input())
c = int(input())
n = int(input())
start = int(input())

if start == 2:
    a, b, c = b, c, a
elif start == 3:
    a, b, c = c, a, b

prices = [a, b, c]
ans = 0
for i in range(n):
    ans = ans + prices[i%3]
print(ans)
```

.

Задача 3. Игра

В данной задаче, вместо самого числа нам требуется его позиция среди всех чисел, ещё не стёртых с доски (вначале позиция числа равна самому числу). Алгоритм решения задачи следующий: пока на доске записано больше одного числа, в зависимости от чётности текущей позиции числа, стираем либо все чётные, либо все нечётные числа, и пересчитываем позицию числа, а также количество оставшихся чисел. Очевидно, что данный алгоритм корректен, ведь по сути он является симуляцией игрового процесса.

Также несложно заметить, что за один ход мы уменьшаем количество чисел в два раза, а значит асимптотическая сложность программы будет равна $O(\log N)$.

```
n = int(input())
position = int(input())
while n > 1:
    if position % 2 == 1:
        print(2)
        n = (n + 1) // 2
        position = (position + 1) // 2
    else:
        print(1)
        n = n // 2
```

```
        position = position // 2
```

Чтобы набрать баллы за тесты $3-10$, можно написать полностью симуляционное решение поместив числа, записанные на доске, в массив (список) и явно удаляя их из массива при каждой операции стирания. Такое решение будет иметь сложность $O(N)$.

```
n = int(input())
position = int(input())
numbers_on_board = list(range(1, n + 1))
while len(lst) > 1:
    i = numbers_on_board.index(position)
    if i % 2 == 0:
        print(2)
        del numbers_on_board[1::2]
    else:
        print(1)
        del numbers_on_board[::2]
```

Для прохождения 11 теста достаточно на каждом ходу стирать числа на чётных позициях.

Для прохождения тестов $12-14$ достаточно заметить, что чётность стираемых на каждом ходу чисел должна отличаться от чётности оставшегося количества чисел. Решение для данной группы тестов могло натолкнуть участников на идею полного решения.

Задача 4. Марсоход

Считаем высоты интересующего участка в массив $h[1..n]$.

Рассмотрим две соседние точки: $h[i]$ и $h[i+1]$. Покажем, что энергия марсохода при перемещении от точки i к точке $i+1$ изменится на $h[i]-h[i+1]$. При положительном значении разности, то есть если $h[i] > h[i+1]$, энергия возрастает на эту величину, так как робот перемещается от более высокой точки к более низкой, аккумулируя при этом единицу энергии за каждую единицу высоты. При отрицательном значении, верно $h[i] < h[i+1]$, а значит робот поднимается, и его энергия уменьшается на единицу за каждую единицу высоты, то есть уменьшается на $h[i+1]-h[i]$, а значит меняется на $-(h[i+1]-h[i]) = h[i]-h[i+1]$. Случай $h[i] = h[i+1]$ не меняет энергию.

Теперь отметим, что перемещение от точки i до точки $i+k$ — это перемещение от i-й точки до $i+1$, от $i+1$ до $i+2$, и так далее до перемещения между точками $i+k-1$ и $i+k$. А значит итоговое изменение энергии считается следующим образом: $(h[i]-h[i+1])+(h[i+1]-h[i+2])+\ldots+(h[i+k-1]-h[i+k])$. Раскроем скобки и обратим внимание, что почти все слагаемые сокращаются, и остается только выражение $h[i]-h[i+k]$.

Значит, изменения заряда можно посчитать по формуле, зависящей от стартовой позиции. Тогда остается только перебрать стартовую позицию циклом и найти l — номер позиции, на котором изменение энергии максимально. При реализации переменную l можно завести явно, и принять изначально равной первой точке, а в теле цикла, на i-м шаге, менять l, если изменение энергии с i-й позиции больше, чем с l-й. Так, улучшая ответ, рассмотрим все возможные стартовые позиции (обратите внимание, что цикл должен рассмотреть только точки $1, 2, \ldots, n-k$), и тогда после завершения последней итерации, l гарантировано содержит точку с наибольшим изменением энергии.

Решение на языке Python приведено ниже. Обратите внимание, что на языке Python массивы нумеруются с нуля, поэтому l инициализируется нулем и по завершению цикла содержит ответ в нумерации с нуля, а значит в ответе нужно вывести $l+1$.

```
n = int(input())
k = int(input())
a = []
for i in range(n):
    a += [int(input())]
l = 0
```

```
for i in range(n - k):
    if a[i] - a[i + k] > a[l] - a[l + k]:
        l = i
print(l+1)
```

Задача 5. Древнее имя

Для решения данной задачи удобно будет сделать массив счётчиков, с помощью которого подсчитывать количество вхождений букв в имя. Соответственно, перебирая буквы имени слева направо, для каждой буквы необходимо увеличивать количество её вхождений в массиве счетчиков, и прибавлять к ответу количество вхождений букв, лексикографически меньших рассматриваемой буквы. Данное решение будет иметь сложность $O(N \cdot |A|)$, где $|A|$ — мощность алфавита, т.е. 26.

```
n = int(input())
name = input()
answer = 0
count_of_occurrences = [0] * 26
for letter in name:
    num_letter = ord(letter) - ord('a')
    count_of_occurrences[num_letter] += 1
    for i in range(num_letter):
        answer += count_of_occurrences[i]
print(answer)
```

Также можно было заметить, что ответом является количество инверсий (только с учётом того, что подразумевается сортировка букв в обратном лексикографическом порядке). Подсчёт количества инверсий можно осуществить с помощью сортировки слиянием, такое решение будет иметь сложность $O(N \log N)$.

Для прохождения тестов $2-11$ достаточно написать переборное решение, которое будет перебирать все пары букв имени, где индекс первой буквы меньше индекса второй буквы. Данное решение будет иметь сложность $O(N^2)$.

```
n = int(input())
name = input()
answer = 0
for i in range(n):
    for j in range(i + 1, n):
        if name[i] < name[j]:
            answer += 1
print(answer)
```

Для прохождения тестов $12-16$ можно было написать решение, которое бы подсчитывало количество вхождений в имя буквы «a» и прибавляло это количество к ответу каждый раз, когда встречало бы букву «b». Данное решение будет иметь сложность $O(N)$.

```
n = int(input())
name = input()
answer = 0
count_a = 0
for letter in name:
    if letter == 'a':
        count_a += 1
    else:
```

```
        answer += count_a
print(answer)
```

Для прохождения тестов 17−21 можно было написать решение, похожее на решение предыдущей группы тестов. А именно, решение должно было, перебирая буквы имени, подсчитывать сколько раз встречались буквы лексикографически меньшие текущей, а также, сколько раз встречались буквы равные текущей. Каждый раз, когда в имени встречается новая буква, необходимо увеличивать количество букв, меньших текущей, на число равное количеству вхождений предыдущей буквы имени. Тогда, для каждой буквы имени, можно будет прибавлять к ответу количество букв меньших данной. Данное решение будет иметь сложность $O(N)$.

```
n = int(input())
name = input()
answer = 0
smaller_letters = 0
current_letters = 1
for i in range(1, n):
    if name[i] != name[i - 1]:
        smaller_letters += current_letters
        current_letters = 1
    else:
        current_letters += 1
    answer += smaller_letters
print(answer)
```