Разбор задач

Задача 1. Кубики

Посмотрим на кубики, которые принесла Валя. Допустим, она достала свой второй кубик — пластмассовый большой синий. Но такого кубика нет у Тани, потому что её первый кубик деревянный, а не пластмассовый, а второй — красный, а не синий. Значит Валя достала свой первый кубик, и он **маленький**.

Посмотрим на кубики, которые принесла Таня. Мы уже выяснили, что общий для всех трех девочек кубик — маленький, а второй кубик Тани большой. Значит, Таня достала первый кубик, и он **деревянный**.

Посмотрим на кубики, которые принесла Маша. Мы уже выяснили, что общий для всех трех девочек кубик — деревянный и маленький. Значит, Маша не могла достать свой первый кубик, потому что он пластмассовый, и не могла достать второй кубик, потому что он большой. Поэтому Маша достала третий кубик, и он деревянный **красный**.

Получается, что у всех трех девочек есть деревянный маленький красный кубик.

Ответ: ДМК

Задача 2. Счастливые билеты

Ближайшие счастливые билеты, большие заданных номеров:

826187
867399
282039
200002
305008

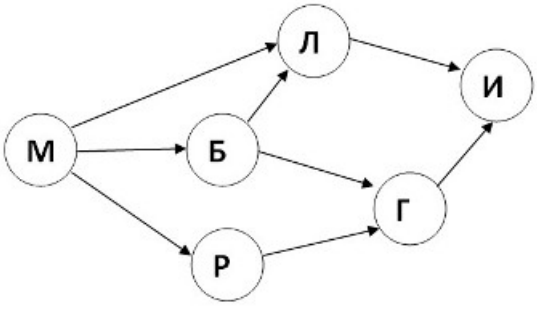
Задача 3. Карточки

Кратчайшее решение содержит 6 строк. Один из вариантов кратчайшего решения:

0100010
1010010
1101010
1110110
1111000
1111111

Задача 4. Домашнее задание

Требования Маши, касающиеся порядка выполнения домашних заданий, можно изобразить в виде схемы. Стрелка означает, что один предмет Маша хочет выучить раньше другого.



Схеме соответствуют следующие планы:

МБЛРГИ
МБРЛГИ
МБРГЛИ
МРБГЛИ
МРБЛГИ

Задача 5. Шифровка

Рассмотрим слово ЕЛЬ. Его шифр 61330 можно разбить на номера букв в алфавите 2 способами:
6, 1, 3, 30
6, 13, 30

Слово ЖАБА зашифровывается 8121. В этом шифре нельзя объединять в пару соседние цифры 8 и 1, потому что буквы с номеров 81 не существует. Поэтому нужно рассмотреть только разбиение фрагмента 121 на номера букв. Есть 3 таких разбиения:
8, 1, 2, 1
8, 12, 1
8, 1, 21

Слово ЛАК имеет шифр 13112. Особенностью этого шифра является то, что любую пару соседних цифр можно объединить, получив номер некоторой буквы. Всего существует 8 способов разбиения этого шифра на номера букв:
1, 3, 1, 1, 2
1, 3, 1, 12
1, 3, 11, 2
1, 31, 1, 2
1, 31, 12
13, 1, 1, 2
13, 1, 12
13, 12, 1

Слово КРУГ зашифровывается 1218214. В этом шифре есть цифра 8, которую нельзя объединить в пару со стоящей справа от неё 2, потому что буквы с номером 82 в русском алфавите нет. Поэтому можно посчитать отдельно, сколько есть способов расшифровать фрагмент 1218 и сколько есть способов расшифровать 214.

Для 1218 получаем 5 способов:
1, 2, 1, 8
1, 2, 18
1, 21, 8
12, 1, 8
12, 18

Для 214 есть 3 способа:
2, 1, 4
2, 14
21, 4

Так как эти фрагменты расшифровываются независимо друг от друга, то к каждому способу расшифровки левого фрагмента можно приписать каждый способ расшифровки правого фрагмента, поэтому для получения ответа на задачу надо умножить полученные числа (5*3). Получим 15 способов.

Слово ТРАМВАЙ имеет шифр 20181143111. Этот шифр содержит цифру 0, которая обязательно должна быть объединена со стоящей слева цифрой 2 для получения номера 20. То есть фрагмент 20 расшифровывается единственным способом, независимо от остальной части шифра.

Далее содержится фрагмент 18, который расшифровывается 2 способами (1, 8 и 18) и тоже не может быть объединен с другими цифрами, потому что номера 81 нет в алфавите.

Следом идет фрагмент 114. Так как буквы с номером 43 нет в алфавите, то этот фрагмент также расшифровывается отдельно от остальных букв. Он имеет 3 способа расшифровки:
1, 1, 4
1, 14
11, 4

И последний фрагмент 3111 можно разбить на номера букв 5 способами:
3, 1, 1, 1
3, 1, 11
3, 11, 1

31, 1, 1
31, 11

Как и в предыдущем задании, к каждому способу расшифровки одного фрагмента можно приписывать каждый способ расшифровки другого фрагмента, поэтому для получения окончательно ответа надо перемножить все полученные способы: 1*2*3*5 = 30

Ответы на задания:
2
3
8
15
30

Разбор задач

Задача 1. Крепость

Сложим количество охранников на всех сторонах, их bn. При этом охранников на башнях мы посчитали два раза, поэтому нужно вычесть an. Получится выражение $bn - an$, которое можно записать в виде `b * n - a * n`, но для того, чтобы минимизировать число операций, ответ должен быть таким:

```
(b - a) * n
```

Задача 2. Счастливые билеты

Ближайшие меньшие и ближайшие большие счастливые билеты:
826178 826187
866992 867399
281920 282039
199991 200002
304700 305008

Задача 3. Сортировка

Кратчайшее решение состоит из 5 строк:
231564
265134
156234
154326
123456

Задача 4. Час расплаты

Первые три числа можно разложить на суммы из 3 слагаемых. Последние два числа раскладываются на суммы из 4 слагаемых. Один из вариантов решения:
21 4 4
21 13 4
21 21 4
22 22 13 22
50 22 13 22

Задача 5. 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 6. 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 7. 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.

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)
```