

## Задача А. Место в поезде

Для решения задачи требуется сначала посчитать количество мест в вагоне  $4 \cdot k$ . Теперь найдем в каком вагоне едет Азат.

```
1 v = 0
2 while n > 4 * k:
3     n -= 4*k
4     v += 1
5 v += 1
```

Теперь найдем в какой кабинке.

```
1 while n > 4:
2     n -= 4
3     cb += 1
4 cb += 1
```

Осталось написать условие для определения верхнее или нижнее место в поезде.

```
1 if n == 1 or n == 4:
2     print("Lower")
3 else:
4     print("Upper")
```

Данный подход набирал 30 баллов. Для полного решения надо избавиться от циклов.

Запись  $\lfloor x \rfloor$  обозначает округление вниз значения  $x$  (например,  $\lfloor 1.4 \rfloor = 1$ ,  $\lfloor 2 \rfloor = 2$ ).

Вычислять в каком вагоне едет Азат можно с помощью формулы:  $v = \lfloor \frac{n+4 \cdot k-1}{4 \cdot k} \rfloor$ . Надо не забыть изменить  $n = n - 4 \cdot k \cdot (v - 1)$ .

Номер кабинки можно вычислить с помощью формулы:  $cb = \lfloor \frac{n+3}{4} \rfloor$ . Надо не забыть изменить  $n = n - 4 \cdot (cb - 1)$ .

Теперь решение имеет асимптотику  $O(1)$ .

## Задача В. Сложные строки

Заметим, что подстроки, которые подходят это палиндромы. Теперь сожмем строку, то есть подстроки с одинаковыми символами заменяем на пару из символа и количество подряд идущих одинаковых символов. То есть из строки *abbbaaabbb* делаем массив *comp* = [(a, 1), (b, 3), (a, 3), (b, 3)]. Теперь осталось пройтись по этому массиву и проверить  $comp[i-1][0] == comp[i+1][0]$ , если данное условие выполняется то проверяем длиннее ли новая подстрока чем ответ, который уже имеется.

```
1 s = input()
2
3 comp = [[s[0], 1]]
4
5 ans = 0
6 for i in range(1, len(s)):
7     if s[i] == comp[-1][0]:
8         comp[-1][1] += 1
9     else:
10        comp.append([s[i], 1])
11
12 for i in range(1, len(comp) - 1):
13     if comp[i-1][0] == comp[i+1][0]:
14         ans = max(ans, comp[i-1][1] + comp[i][1] + comp[i+1][1])
15 print(ans)
```

## Задача С. Рассадка пассажиров

### Подзадача 1

В текущей подзадаче можно создать двумерный массив  $a$  размером  $n$  на 3 и эмулировать процесс рассадки. Значение  $a[i][j]$  равное *true* будет означать, что в ряду  $i$  на кресле  $j$  уже сидит пассажир, иначе кресло ещё пустое. При посадке очередного пассажира нужно проверить: сидит ли кто-нибудь

между проходом и креслом пассажира, и если на указанных местах есть какие-либо пассажиры, то нужно увеличить ответ на количество таких пассажиров. Асимптотика решения  $O(k \cdot m)$ , а так как  $m$  — это константа, то имеем асимптотику  $O(k)$ . Количество пассажиров не больше общего количества мест, поэтому  $k \leq n \cdot m$  и итоговая асимптотика решения  $O(n)$ .

## Подзадача 2

Текущую подзадачу можно решить аналогично подзадаче 1, но нужно создать массив размером  $n$  на  $m$ . Различие лишь в том, что в подзадаче 1 для каждого места в ряду можно было точно проверить остальные места в том же ряду, а в текущей подзадаче лучше использовать цикл. Асимптотика решения  $O(n \cdot m^2)$ .

## Подзадача 3

В текущей подзадаче можно отсортировать массив по номеру ряда, при этом для одного и того же ряда нужно сохранить порядок пассажиров, который был в исходном списке. Другими словами, если пассажиры  $i$  и  $j$  имеют кресла на одном ряду и в изначальном списке пассажир  $i$  идёт до пассажира  $j$ , то и после сортировки пассажир  $i$  должен идти до пассажира  $j$ .

Так как мы отсортировали пассажиров по номеру ряда, то мы можем пойти по отсортированному массиву и обрабатывать ряд за рядом, для каждого ряда эмулируя процесс посадки. То есть в отличие от предыдущих подзадач можно создать одномерный массив размером  $m$ .

Итоговая асимптотика решения  $O(k \cdot \log k) + O(k \cdot m) = O(k \cdot (\log k + m))$ .

## Подзадача 4

Решение основывается на решении из подзадачи 3, но теперь нужно обрабатывать каждого пассажира быстрее чем  $O(m)$ . Для этого можно воспользоваться, например, деревом отрезков и при обработке очередного пассажира вычислять количество пассажиров на отрезке в ряду за  $O(\log m)$ . В таком случае итоговая асимптотика будет  $O(k \cdot \log k) + O(k \cdot \log m) = O(k \cdot (\log k + \log m))$ .

## Подзадача 5

Заметим, что задача сводится к задаче о подсчёте количества инверсий. В таком случае нам не нужно хранить данные о текущем обрабатываемом ряду в одномерном массиве размером  $m$ , а можно обойтись массивом размером, равном количеству пассажиров в текущем ряду. Асимптотика решения  $O(k \cdot \log k)$ .

## Задача D. Максимальное покрытие

Для решения задачи на 30 баллов достаточно перебрать начало первого отрезка, после чего найти минимальное значение в каждом отрезке, взять максимум по посчитанным минимумам и выбрать наилучший вариант из всех способов начать первый отрезок.

Далее обсудим решение третьей подзадачи. В ней  $n$  делится на  $k$ . Это значит, что достаточно начало первого отрезка можно перебирать от 0 до  $k - 1$ , и, поскольку всего отрезков будет  $n/k$ , то вычислить значение искомой функции не составит труда, заранее предподсчитав минимум в каждом из отрезков длины  $k$ . Асимптотика такого решения  $O(n \log n)$  из-за предподсчета.

Для решения на полный балл заметим используется следующая идея. Давайте переберем позицию  $p$  короткого отрезка, если такой есть (то есть если  $n$  не делится на  $k$ ). Теперь давайте заметим, что если переместить отрезок с позиции  $p$  на позицию  $p + k$ , то большинство отрезков останутся без изменений, а значит такой перенос можно обработать за  $O(\log n)$ , поддерживая текущие значения минимумов внутри каждого отрезка. Тогда если  $p \bmod k$  фиксированно, то можно обработать все такие позиции  $p$  за  $O(n/k \log n)$ . Тогда суммарно алгоритм работает за  $O(n \log n)$ .

## Задача E. Постройка домов

## Подзадача 1

Так как количество различных чисел в массиве равно двум и  $t = 1$ , задача сводится к решению линейного диофантового уравнения:  $brick_0 \cdot cnt_0 + brick_1 \cdot cnt_1 = q$ , где  $brick_0$  и  $brick_1$  - уникальное количество кирпичей на складах, а  $cnt_i$  - количество нужных кирпичей  $brick_i$ . Посчитаем  $brick_0$  и  $brick_1$  с помощью словаря и переберем все возможные варианты. Асимптотика решения  $O(n^2)$

## Подзадача 2

Для решения подзадачи 2 достаточно рекурсивно перебрать все возможные суммы кирпичей со складов и для каждой суммы также рекурсивно перебрать сумму кирпичей для заказов, проверяя равны ли суммы. Если суммы оказались равны максимизируем ответ. Асимптотика решения  $O(2^{n+t})$

```
1 n = int(input())
2 brick = list(map(int, input().split()))
3 t = int(input())
4 query = list(map(int, input().split()))
5 ans = 0
6
7 def f_query(t, sum_query, k, sum_brick, cnt):
8     if (k == t):
9         global ans
10        if (sum_query == sum_brick):
11            ans = max(ans, cnt)
12        return
13        f_query(t, sum_query + query[k], k + 1, sum_brick, cnt + 1)
14        f_query(t, sum_query, k + 1, sum_brick, cnt)
15
16 def f_brick(n, sum_brick, k):
17     if (k == n):
18         global t
19         if (sum_brick > 0):
20             f_query(t, 0, 0, sum_brick, 0)
21         return
22         f_brick(n, sum_brick + brick[k], k + 1)
23         f_brick(n, sum_brick, k + 1)
24
25 f_brick(n, 0, 0)
26 print(ans)
```

## Подзадача 3

Для решения подзадачи 3 необходимо улучшить рекурсивный перебор, сохраняя значения в массиве  $dp\_brick$  и  $dp\_query$ . Массив  $dp\_brick$  будем заполнять по следующему правилу:  $dp\_brick[i] = 1$ , если сумму  $i$  можно собрать используя какие-то кирпичи со складов

```
1 n = int(input())
2 brick = list(map(int, input().split()))
3 t = int(input())
4 query = list(map(int, input().split()))
5 W = 10000 + 1
6 dp_brick = [0] * W
7 dp_query = [0] * W
8 ans = 0
9 def f_tree(n, sum_brick, k):
10    if (k == n):
11        dp_brick[sum_brick] = 1
12        return
13        f_tree(n, sum_brick + brick[k], k + 1)
14        f_tree(n, sum_brick, k + 1)
15
16 def f_query(t, sum_query, cnt, k):
17    if (k == t):
```

```
18     dp_query[sum_query] = max(dp_query[sum_query], cnt)
19     if (dp_brick[sum_query]):
20         global ans
21         ans = max(ans, dp_query[sum_query])
22     return
23     f_query(t, sum_query + query[k], cnt + 1, k + 1)
24     f_query(t, sum_query, cnt, k + 1)
25
26 f_tree(n, 0, 0)
27 f_query(t, 0, 0, 0)
28 print(ans)
```

#### Подзадача 4

Данная подзадача подразумевала за собой использование алгоритма «укладки рюкзака».  $dp\_brick[i][j] = 1$ , если сумму  $j$  можно собрать используя какие-то первые  $i$  кирпичей со складов из нашего массива  $brick$ . Соответственно пересчет нашей динамики будет:  $dp\_brick[i][j] = dp\_brick[i - 1][j]$  or  $dp\_brick[i - 1][j - brick[i]]$ . Так как  $t = 1$ , в  $dp\_brick[n][query_0]$  будет лежать наш ответ, где  $query_0$  - это наш единственный заказ. Асимптотика решения  $O(n \cdot W)$ .

Пересчет динамики:

```
1 for i in range(1, n + 1):
2     for w in range(W):
3         if (w - brick[i] >= 0):
4             dp_brick[i][w] = dp_brick[i - 1][w - brick[i]] or dp_brick[i - 1][w]
5         else:
6             dp_brick[i][w] = dp_brick[i - 1][w]
```

#### Подзадача 5

Для решения этой подзадачи необходимо воспользоваться алгоритмом «укладки рюкзака со стоимостью». Соответственно  $dp\_query[i][j]$  - максимальное количество заказов, которые мы можем выбрать и выполнить, чтобы набрать сумму  $j$ , используя какие-то первые  $i$  заказы. Единственное изменение алгоритма будет в пересчете динамики, так как нам необходимо максимизировать количество заказов, каждый раз жадно берем еще один заказ, если можем:

$$dp\_query[i][j] = \max(dp\_query[i - 1][j], dp\_query[i - 1][j - query[i]] + 1)$$

Асимптотика решения  $O(t \cdot W)$ .

Пересчет динамики:

```
1 for i in range(1, t + 1):
2     for w in range(W):
3         if (w - query[i] >= 0):
4             dp_query[i][w] = max(dp_query[i - 1][w], dp_query[i - 1][w - query[i]] + 1)
5         else:
6             dp_query[i][w] = dp_query[i - 1][w]
```

#### Подзадача 6

Для решения задачи на полный балл необходимо объединить решения подзадач 4 и 5. Для вычисления ответа пройтись по всем возможным суммам, если сумму  $i$  можно собрать используя заказы древесины и эту же сумму можно набрать, используя длины деревьев, максимизируем ответ. Асимптотика решения  $O(W \cdot (n + t))$ .

```
1 n = int(input())
2 brick = list(map(int, input().split()))
3 t = int(input())
4 query = list(map(int, input().split()))
5 brick = [0] + brick
```

```
6 query = [0] + query
7 W = 10001
8
9 dp_brick = [[0 for row in range(0, W)] for col in range(0, n + 1)]
10 dp_brick[0][0] = 1
11
12 dp_query = [[-W for row in range(0, W)] for col in range(0, t + 1)]
13 dp_query[0][0] = 0
14
15 for i in range(1, n + 1):
16     for w in range(W):
17         if (w - brick[i] >= 0):
18             dp_brick[i][w] = dp_brick[i - 1][w - brick[i]] or dp_brick[i - 1][w]
19         else:
20             dp_brick[i][w] = dp_brick[i - 1][w]
21
22
23 for i in range(1, t + 1):
24     for w in range(W):
25         if (w - query[i] >= 0):
26             dp_query[i][w] = max(dp_query[i - 1][w], dp_query[i - 1][w - query[i]] + 1)
27         else:
28             dp_query[i][w] = dp_query[i - 1][w]
29 ans = 0
30 for i in range(W):
31     if (dp_brick[n][i] and dp_query[t][i]):
32         ans = max(ans, dp_query[t][i])
33 print(ans)
```