

Задача А. Почти $a + b$

Разработчик задачи: Мария Ткаченко

Заметим, что ответ на задачу равен $(a - x + 1) \cdot (b - x + 1)$. Это связано с тем, что левый верхний угол квадрата мы можем разместить в любой клетке внутри прямоугольника $(a - x + 1) \times (b - x + 1)$. Решение можно реализовать за $\mathcal{O}(1)$.

Задача В. Какой сегодня день?

Разработчик задачи: Мария Левина

Заметим, что день $x + y$ можно представить в виде $x + y = k \cdot \text{month} + t$, где k — количество полных прошедших месяцев; $\text{month} = a + b + c + d$ — количество дней в месяце; t — число текущего месяца.

Текущий день считается ещё не прошедшим, поэтому для вычисления k мы делим нацело $(x + y - 1)$ прошедших дней на month . Вычислив значение k , мы сможем определить $t = x + y - k \cdot \text{month}$. Теперь мы знаем, какой номер у нужного нам дня t .

У дня t первая неделя, если $t \leq a$; вторая — если $a < t \leq a + b$; третья — если $a + b < t \leq a + b + c$; иначе — четвёртая.

Задача С. Чередующаяся чётность

Для решения задачи нам нужно найти все цифры в записи числа x . Для этого мы можем сделать следующее:

- Можно рассмотреть число как строку. Для этого достаточно его сразу именно так и считать.
- Можно разложить число на разряды. Для этого можно делать следующее:
 - Пока число не станет равным нулю, будем находить его последнюю цифру (для этого можно применить операцию взятия остатка по модулю 10) и убрать её из числа (для этого можно разделить число целочисленно на 10).

Задача D. Олимпиада КоРОЧе

Автор задачи: Алексей Егоров. Разработчик задачи: Михаил Грибакин

Для каждого участника определим его место на олимпиаде отдельно. Для этого нам нужно будет найти количество участников, у которых решено строго больше задач. Другими словами, нам нужно считать массив a из задачи и для каждого индекса i найти количество индексов $1 \leq j \leq n$, что $a_j > a_i$.

Решение можно реализовать за $\mathcal{O}(n^2)$.

Задача Е. Волшебное зеркало

Разработчик задачи: Анастасия Корюкина

Для решения задачи создадим массив из 10 элементов, где для каждой цифры i будем хранить её отражение (если оно существует), а для недопустимых цифр будем хранить -1 . Затем для каждой цифры в строке проверим, совпадает ли она с отражением соответствующей ей цифры с конца строки. Если все такие пары совпадают, то строка красивая.

Решение можно реализовать за $\mathcal{O}(|s|)$.

Задача F. Размещение людей

Разработчик задачи: Мария Потапова

В этой задаче можно для каждого пассажира выбирать место полным перебором. Для этого переберём все варианты и за $\mathcal{O}(n)$ найдём минимальное расстояние до уже занятого места.

Решение можно реализовать за $\mathcal{O}(n^3)$.

Задача Г. Просто считаем квадраты

Разработчик задачи: Егор Зарубин

Давайте задачу на отрезке разобьём на две задачи на префиксе. Другими словами, пусть $f(n)$ — количество квадратов, которые не больше n . Тогда ответ на задачу будет равен $f(r) - f(l - 1)$.

Научимся считать $f(n)$. Для этого заметим, что нам подойдут все такие значения x , что $0 \leq x \leq \sqrt{n}$. Другими словами, $f(n) = \lfloor \sqrt{n} \rfloor + 1$.

В задаче вам нужно достаточно аккуратно извлекать корень и округлять его вниз, чтобы не было проблем с точностью. Также нужно отдельно рассмотреть случай вызова функции $f(-1)$.

Решение можно реализовать за $\mathcal{O}(1)$. Также задачу можно было решать с помощью бинарного поиска.

Задача Н. Число и его квадрат

Разработчик задачи: Максим Сименков

Для решения задачи можно использовать структуру данных множество. Сделаем следующее:

- Давайте все числа из массива a сложим в множество.
- Будем перебирать число x из множества, и если в множестве также есть число x^2 , то увеличим ответ на единицу.

Решение можно реализовать за $\mathcal{O}(n \log n)$.

Задача І. Червячки

Автор и разработчик задачи: Александр Волков

Будем решать задачу жадно:

- Пройдём по всем i от 0 до $n - 2$ слева направо.
- Если $a_i \neq b_i$, то в массиве b найдем нужный элемент среди $b_{i+1}, \dots, b_{n-1}$ и поменяем его местами с b_i . Другими словами, найдём такой индекс $i < j < n$, что $b_j = a_i$ и поменяем местами b_i и b_j .

Таким образом, чтобы поставить один элемент на свое место, нам потребуется не более одного обмена, а значит на весь массив потребуется не более n обменов. Решение можно реализовать за $\mathcal{O}(n^2)$.

Задача Ј. Количество делителей

Разработчик задачи: Тимофей Венидиктов

Количество делителей произведения двух взаимно простых чисел равно произведению количеств их делителей. Иными словами, если обозначить за $f(x)$ — количество делителей числа x , то $f(x \cdot y) = f(x) \cdot f(y)$ для взаимно простых чисел x и y . Это связано с тем, что каждый делитель их произведения уникально образуется комбинацией делителей первого и второго чисел, а отсутствие общих множителей гарантирует, что каждая комбинация уникальна.

Найти количество делителей числа x можно за $\mathcal{O}(\sqrt{x})$. Решение можно реализовать за $\mathcal{O}(\sqrt{x} + \sqrt{y})$.

Задача К. Системы счисления

Автор задачи: Михаил Горбатов. Разработчик задачи: Прохор Иголкин

Для решения задачи нам потребуется сделать следующее:

- Перевести число a в десятичную систему счисления. Аналогично перевести число b в десятичную систему счисления.
- В десятичной системе счисления мы можем найти сумму и произведение чисел.
- Далее нам нужно два полученных числа перевести в систему счисления с основанием e .

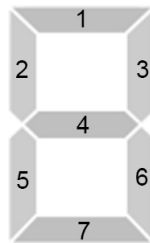
Задача L. Электронные часы

Автор и разработчик задачи: Никита Насонков

Это задача на реализацию, поэтому её можно решать разными способами. Рассмотрим здесь один из них.

Вкратце можно определить, какие цифры могут стоять на каждой из 4 позиций, а затем перебрать все их комбинации, выбрав из них те, что являются валидными значениями времени.

Сначала нужно научиться определять, может ли конкретная цифра x стоять на конкретной позиции на циферблате, исходя из видимого Поликарпу набора сегментов. Для этого пронумеруем сегменты, как показано на рисунке:



Затем для позиции на циферблате сделаем троичную маску `position_mask` длины 7, в которой `position_mask[i]` будет равно 0, если i -й сегмент (в единичной индексации) не горит, 1 — если i -й сегмент горит, и 2, если он скрыт от зрения Поликарпа. Например, маска 0121122 будет соответствовать следующей конфигурации сегментов на позиции для цифры на циферблате:

```
.....  
1....X  
1....X  
1....X  
.1111.  
1....X  
1....X  
1....X  
.XXXX.
```

Для каждой из цифр от 0 до 9 также сделаем маску `digit_mask` длины 7 по похожему принципу — `digit_mask[i]` будет равно 1, если i -й сегмент в отображении этой цифры на циферблате горит, и 0, если он не горит. Тогда для цифр по порядку от 0 до 9 включительно соответствующие маски получатся 1110111, 0010010, 1011101, 1011011, 0111010, 1101011, 1101111, 1010010, 1111111 и 1111011.

Тогда чтобы проверить, что конкретная цифра может стоять на конкретной позиции, нужно проверить, что для каждого i от 1 до 7 `position_mask[i]` либо равно 2, либо равно `digit_mask[i]`.

Сделав такую проверку для каждой пары из цифры и позиции, мы получим 4 массива — наборы цифр, которые могут стоять на каждой из 4 позиций. Затем можно перебрать все возможные комбинации значений на позициях, используя 4 вложенных цикла `for`. Каждая комбинация — это 4 цифры a , b , c и d — цифры десятков часов, единиц часов, десятков минут и единиц минут соответственно. Чтобы проверить, что запись `ab:cd` является валидным моментом времени на циферблате, необходимо, чтобы выполнялись два неравенства — $a \cdot 10 + b \leq 23$ и $c \cdot 10 + d \leq 59$. Если оба неравенства выполняются, то запись `ab:cd` можно добавлять в массив с ответом.

В конце массив с ответом сортируем лексикографически, затем выводим его длину и содержимое, в указанном в условии формате. Заметим, что если часы сломаны, то массив с ответом получится длины 0, а значит, в результате действий, описанных выше, будет выведен как раз 0 в единственной строке, как и просят в условии, поэтому никакой отдельной проверки на сломанность часов делать не придётся.

Задача М. Закупаемся в магазине

Автор и разработчик задачи: Владимир Герасиков

Будем решать задачу методом двух указателей. Давайте для каждой левой границы l найдём наибольшую правую границу r_l , что мы можем купить все товары с номерами $l, l+1, \dots, r_l$, потратив не более s рублей. Заметим, что эти наибольшие правые границы r_l не убывают.

Чтобы реализовать два указателя в этой задаче, нам нужно будет поддерживать сумму стоимостей товаров в нашем текущем окне (между нашими указателями), а также в мультимножестве поддерживать стоимости товаров. Мультимножество нам потребуется для того, чтобы быстро определять максимальное значение в окне.

Ответом на задачу будет максимальное значение $r_l - l + 1$ по всем значениям l . Решение можно реализовать за $\mathcal{O}(n \log n)$.

Задача N. Дивергенция

Автор задачи: Арсений Ятчений. Разработчик задачи: Глеб Костылев

Напишем динамику по поддеревьям: $dp[v][set]$ — длина кратчайшего пути от v до некоторой вершины u в поддереве v , такого, что множество мощностей D-mail-ов на этом пути равно set . Так как в множестве мощностей может быть только три элемента: 0, 1 и 2, есть всего 2^3 возможных множеств set , которые нам интересно хранить.

Научимся пересчитывать $dp[v]$. Если v — это лист, то множество мощностей на пути может быть только пустым, поэтому нам нужно сохранить только $dp[v][\emptyset] = 0$. Если же v — не лист, то переберём u_i — детей v , насчитаем $dp[u_i]$ рекурсивно, после чего переберём все set , сохранённые в $dp[u_i]$. Напомним, что в $dp[u_i]$ для каждого множества хранится кратчайший путь от u_i до какой-то вершины, на котором есть D-mail-ы из этого множества, а в $dp[v]$ нам нужен путь от v . Продлим эти пути из $dp[u_i]$ до v : добавим в set элемент a_{u_i} и пересчитаем $dp[v][set \cup \{a_{u_i}\}]$ через $dp[u_i][set] + 1$.

После того как все dp насчитаны, нужно посчитать ответ. Для пересчёта ответа нам понадобится массив $ans[set]$, в котором будет храниться кратчайшая длина пути, на котором есть все мощности из множества set . Для начала обновим $ans[set]$ через все $dp[v][set]$. Однако в $dp[v]$ мы рассматривали только вертикальные пути: те, которые спускаются из v в поддерево v . Для нахождения ответа нам нужно рассмотреть вообще все пути в дереве, в том числе и такие, которые стартуют в v , поднимаются вверх в некоторую вершину w , а потом спускаются из w вниз в вершину u . Переберём w , обозначим её детей за s и t . Переберём, какое подмножество мощностей set_s мы хотим найти в поддереве s , а какое — set_t — в поддереве t , и пересчитаем $ans[set_s \cup set_t \cup \{a_s\} \cup \{a_t\}]$ через $dp[s][set_s] + dp[t][set_t] + 2$.

Если в ans нашлось множество $\{0, 1, 2\}$, то мы можем вернуться в прошлое на 3 дня при помощи пути длиной $ans[\{0, 1, 2\}]$. Если нашлось $\{0, 1\}$, то, соответственно, на два дня, если $\{0\}$, то на один, а если ни на каком пути нет D-mail-а мощности 0, то ответ — 0.

Решение можно реализовать за $\mathcal{O}(n \cdot 2^6)$.

Задача O. Побег из графа

Автор и разработчик задачи: Илья Богатырев

Построим граф с инвертированными рёбрами и запустим для него алгоритм Дейкстры, вычисляющий в каждой вершине v минимальную сумму денег $d[v]$, с которой из этой вершины можно покинуть граф. Множество стартовых вершин — это вершины с выходами. Величины $d[v]$ для них равны c_v . Все эти вершины нужно сразу поместить в очередь с приоритетами.

Далее при каждом переходе по инвертированному ребру из v в u будем подбирать такое стартовое значение $d[u]$, чтобы $d[u] - \lfloor \sqrt{d[u]} \rfloor \geq d[v]$. В силу монотонности это можно делать бинарным поиском.

Если в качестве очереди с приоритетами использовать C++ set, то общее время работы составит $\mathcal{O}(m \log n + m \log \max)$, где $\max$ — максимальный ответ в вершине, который в худшем случае не превышает 10^{10} . Также эта оценка позволяет заметить, что извлечение квадратного корня при вычислениях можно выполнять с использованием стандартного $\text{sqrt}()$, не прибегая к дополнительным приёмам для повышения точности.

Задача Р. Лексикографическая минимизация

Автор и разработчик задачи: Арсений Ятчений

Данная задача решается с помощью динамического программирования по подмножествам. Каждому индексу i от 1 до n поставим в соответствие подмножество $s_i \subseteq \{1, 2, \dots, m\}$ такое, что объединение всех s_i даёт множество $\{1, 2, \dots, m\}$. Обратите внимание, что мы рассматриваем такие множества s_i , которые попарно не пересекаются. Тогда каждое такое подмножество s_i можно представить в виде битовой маски mask_i , где j -й бит равен 1, если $j \in s_i$, и 0 иначе.

В самом начале для каждого mask предсчитаем $\text{xorb}_{\text{mask}} = b_{i_1} \oplus b_{i_2} \oplus \dots \oplus b_{i_k}$, где на позициях $i_1, i_2, \dots, i_k$ в mask стоят единицы. Так мы сможем быстро отвечать за $\mathcal{O}(1)$ на запрос вычисления хог для подмножества mask . Предсчитать такой массив мы можем за $\mathcal{O}(2^m \cdot m)$.

Будем считать динамику $\text{dp}_{i,\text{mask}}$, в которой будем хранить 0/1. Параметры в динамике будут следующими:

- mask — битовая маска, где j -й бит 1, если b_j был использован, и 0 иначе. Сразу заметим, что $0 \leq \text{mask} < 2^m$.
- i — длина рассмотренного префикса массива a .
- $\text{dp}_{i,\text{mask}} = 1$, если можно набрать минимальный лексикографически массив $a_1, a_2, \dots, a_i$ используя только b_j из множества mask . Если набрать минимальный лексикографически массив нельзя, то $\text{dp}_{i,\text{mask}} = 0$.
- $\text{dp}_{i,\text{mask}} = 0$ иначе.

Пусть мы сейчас обновляем i -й слой динамики, где $1 \leq i \leq n$. Рассмотрим пересчёт:

- Будем перебирать маску mask из чисел, с которыми мы хотим проксорить a_i . Тогда a_i станет равным $a_i \oplus \text{xorb}_{\text{mask}}$.
- Так как наша цель — построить лексикографически минимальный массив длины i , то по индукции предполагаем, что префикс длины $i - 1$ уже является минимальным. Следовательно, для текущей маски mask необходимо найти такую маску prevmask из предыдущего слоя динамики, что $\text{prevmask} \& \text{mask} = 0$ и $\text{dp}_{i-1,\text{prevmask}} = 1$. Если такая предыдущая маска найдется, то сложим в массив canupdate пару $(a_i \oplus \text{xorb}_{\text{mask}}, \text{prevmask} \mid \text{mask})$. В массиве canupdate мы будем хранить состояния (i, mask) , которые мы можем сделать равными 1.
- Так как $\text{prevmask} \& \text{mask} = 0$, то prevmask — это просто подмаска дополнения маски mask . Другими словами, prevmask является подмаской $(2^m - 1) \oplus \text{mask}$. Перебор всех масок размера m и их подмасок — это известная задача с асимптотикой работы $\mathcal{O}(3^m)$.
- Рассмотрим массив достижимых состояний canupdate . Среди таких состояний мы можем сделать равными 1 те и только те, у которых $a_i \oplus \text{xorb}_{\text{mask}}$ минимален, так как мы уже набрали минимальный лексикографически префикс длины $i - 1$ и чтобы лексикографическая минимальность сохранилась, к нему нужно добавить наименьшее число.

Поэтому среди всех пар $(\text{val}, \text{mask})$ из массива canupdate мы сделаем $\text{dp}_{i,\text{mask}} = 1$ только для тех, для которых val минимален.

- Заметим, что в процессе построения динамики мы сразу находим ответ — на каждой итерации значение $minval$ будет очередным элементом из оптимального массива a .
- Обратите внимание, что последний слой динамики нужно рассмотреть отдельно, так как по условию задачи мы должны каждому элементу b_j выбрать некоторое a_i .

Решение можно реализовать за $\mathcal{O}(3^m \cdot n)$.