Задача A. Баланс настроения

Автор задачи и разработчик: Константин Бац

Рассмотрим величину $y_i = x_i + 2i$, где x_i — настроение Кости в i-ю минуту, а i — собственно, минута прогулки. Проследим, как меняется эта величина при переходе от минуты i к минуте $i+1$.

- Если Косте пришла в голову негативная мысль, x_i превращается в $x_{i+1} = x_i - 1$, поэтому $y_{i+1} = (x_i - 1) + 2(i+1) = y_i + 1$;
- Если же Косте пришла в голову неоднозначная мысль, x_i превращается в $x_{i+1} = 2(x_i + (i+1) - 2) = 2(x_i + i - 1)$, то есть $y_{i+1} = 2(x_i + i - 1) + 2(i+1) = 2y_i$.

Таким образом, с каждой следующей минутой прогулки величина y либо увеличивается на один, или увеличивается в два раза. Заметим так же, что изначально, до первой минуты прогулки, $y_0 = 0$, а в конце прогулки мы хотим получить $y_n = 0 + 2n$. Получается, что мы свели задачу к получению числа $2n$ из числа 0 операциями прибавления единицы и умножения на два. Более того, требуется как можно меньше раз использовать операцию, которая дает $y_{i+1} = y_i + 1$, так как требуется минимизировать количество негативных мыслей.

Посмотрим на первый раз, когда будет применена операция увеличения на 1. Заметим, что после этого y_i будет только расти, и при этом

- мы не можем сделать больше $\lfloor \log_2(2n) \rfloor$ умножений на два, потому что тогда конечный y_n будет больше $2n$;
- мы обязаны сделать хотя бы $\texttt{bitcount}(2n)$ операций прибавления единицы, потому что операция умножения на два не увеличивает количество единичных бит в числе.

Из этих двух наблюдений получается алгоритм: будем «собирать» число $2n$ по битам от старших к младшим: каждый раз умножаем текущий результат на два (неоднозначная мысль), и, если соответствующий бит в числе $2n$ равен единице, прибавляем к результату 1 (негативная мысль). Время работы алгоритма — $\mathcal{O}(\log n)$.

Задача B. Приятный плейлист

Автор задачи и разработчик: Даниил Голов

Рассмотрим случай, когда Даниил любит только одну какую-то песню. Тогда он послушает ее k раз и получит удовольствие $a_1 + (a_1 - 1) + (a_i - 2) + \ldots$. Количество прослушиваний песни в плейлисте, когда она приносит хоть какое-то удовольствие, равно $t = \min(k, a_i)$. Поэтому ответом будет число $a_1 + (a_1 - 1) + \ldots + (a_1 - t + 1) = t\left(a_1 - \frac{t-1}{2}\right)$.

Теперь рассмотрим случай, когда $n > 1$. Тогда всегда найдутся две песни, которые можно слушать по очереди и удовольствие от их прослушивания не будет уменьшаться. Выберем две песни с наибольшими a_i, пусть они, не теряя общности, имеют номера 1 и 2 и пусть $a_1 \geqslant a_2$. Рассматривать какие-то песни, кроме этих двух, не имеет смысла. Если Даниил каждый раз выбирает песню с максимальным на текущий момент a_i, то до песен с третьим и ниже по величине a_i он никогда не дойдет.

Обозначим $d = a_1 - a_2$. Если $d = 0$, то есть $a_1 = a_2$, Даниил за каждое прослушивание будет получать ровно a_1 удовольствия, и ответ будет na_1. Иначе, после d прослушиваний первой песни, Даниил получит $a_1 + (a_1 - 1) + \ldots + (a_2 + 1)$ удовольствия, после чего один раз послушает вторую песню вместо первой. В итоге удовольствие от прослушивания первой песни примет свое изначальное значение a_1, и Даниил снова повторит этот цикл.

Таким образом, весь плейлист разбивается на одинаковые блоки по $d+1$ песен, и, возможно, еще часть такого блока в конце. Суммарное удовольствие в блоке или в начале блока можно вычислить по формуле суммы арифметической прогрессии, поэтому асимптотика такого решения — $\mathcal{O}(1)$.

Задача C. Пропал мусор

Автор задачи и разработчик: Владимир Рябчун

Для решения этой задачи удобно рассматривать массив длины 2^k, для этого исходный массив можно дополнить до ближайшей степени двойки нулями в конце.

Ответ на запрос суммы будем выполнять так: для каждого бита нужно посчитать, в каком количестве слагаемых этот бит равен единице. Пусть таких слагаемых c_d. Тогда ответом будет $\sum_{d=0}^{15} 2^d \cdot c_d$. Опишем вычисление величин c_d.

Будем решать задачу отдельно для каждого бита. Для бита d весь массив разобьётся на отрезки длины 2^d, на которых у индексов в массиве этот бит будет постоянным (так, третий бит равен 0 для индексов с 0 по 7, равен 1 для индексов с 8 по 15, и так далее).

Заметим, что c_d состоит из количества элементов массива a_i, имеющих 1 в d-м бите, стоящих на позиции i, имеющей 0 в d-м бите, и наоборот. Если рассматривать только d-й бит, у нас имеется массив из 0 и 1, в котором некоторые элементы нужно рассматривать инвертированными (если в индексе i стоит 1 в d-м бите), а некоторые — нет (если в индексе стоит 0). Воспользуемся свойствами двоичной записи натуральных чисел, а именно:

- все такие отрезки имеют длинну 2^d;
- а благодаря тому, что $n = 2^k$, эти отрезки соответствуют внутренним вершинам обыкновенного дерева отрезков для данного массива.

Тогда построим дерево отрезков на сумму (считать будем количество единиц на отрезке), но не будем опускаться ниже d-го слоя, сделаем вершины, соответствующие отрезкам длины 2^d, листьями. А в каждом из этих листьев построим отдельное дерево отрезков, позволяющее быстро выполнять операции изменения (битовые операции для каждого конкретного бита транслируются в присвоение или инверсию).

Запросы к внешнему дереву отрезков посещают $\mathcal{O}(1)$ листьев, в каждом из которых выполнится запрос за $\mathcal{O}(\log N)$ действий, поэтому асимптотика любого запроса для фиксированного бита будет $\mathcal{O}(\log N)$.

Итоговая асимптотика будет $\mathcal{O}(\log N \cdot \log M)$, где M — максимальное значение элементов в массиве.

Задача D. Осеннее палиндромище

Автор задачи и разработчик: Владислав Власов

Давайте сначала решим более простую задачу: сделать палиндромами все столбцы. Рассмотрим желаемую матрицу s, в ней $s_{1,1} = s_{n,1}$, $s_{1,2} = s_{n,2}$, $\ldots$, $s_{1,m} = s_{n,m}$. Получается, что первая строка должна совпадать с последней, аналогично вторая строка должна совпадать с предпоследней и так далее. Значит каждой строке должна сопоставляться такая же, кроме случая с нечётным n, тогда одна строка не может иметь пары; если более чем у одной строки нет пары, то палиндром составить невозможно.

Теперь соберём палиндром по столбцам. Воспользуемся двусторонней очередью и положим в неё строку без пары. Теперь будем класть в очередь парные: одну слева, другую справа. Для того, чтобы обрабатывать одинаковые строки вместе и в принципе знать количество вхождений каждой строки в матрицу, можно было воспользоваться ассоциативным массивом (`unordered_map`, `HashMap`, `dict`).

Заметим, что мы меняли местами только строки.

Теперь нам осталось сделать палиндромами строки матрицы. Давайте транспонируем её, то есть повернём на 90°. Теперь столбцы стали строками и наоборот. Значит на текущий момент каждая строка уже является палиндромом, и мы умеем делать палиндромами столбцы, не меняя порядок символов в строках, а меняя только строки местами.

Проделаем это снова, если хотя бы один раз было невозможно составить палиндром по столбцам, то ответ «`NO`». Время работы решения — $\mathcal{O}(nm)$.

Задача E. Очерк

Автор задачи и разработчик: Александр Гордеев

Согласно условию, каждую клетку мы можем красить сколько угодно раз. Нам важно чтобы все клетки, которые должны остаться белыми, остались белыми (красить их нельзя), а все клетки, которые должны стать красными, стали красными (необходимо чтобы хоть одна кисть задела эту клетку).

Заведем массив $\texttt{state}_{i,j}$, который хранит состояние каждой клетки. Если $\texttt{state}_{i,j} = 0$, то клетку (i, j) нужно покрасить, если $\texttt{state}_{i,j} = 1$, то её красить нельзя, если $\texttt{state}_{i,j} = 2$, то её мы уже покрасили.

Теперь можно просто пройтись по матрице и попытаться поставить все возможные варианты поставить любую кисть в каждое возможное место. Если все $\texttt{state}_{x,y}$, где (x, y) — клетка, до которой мы достаём нашей текущей кистью, равны 0 или 2, то при таком расположении кисти нет клеток, которые красить запрещено, и мы можем поставить кисть, то есть для всех затронутых клеток присвоить $\texttt{state}_{x,y} = 2$.

После прохода по матрице достаточно проверить, что не осталось не одной клетки, для которой $\texttt{state}_{i,j} = 0$. Действительно, если осталась непокрашенная клетка, то ни одно расположение кисти, красящее её, не было корректным. Таким образом, если не осталось клеток со $\texttt{state}_{i,j} = 0$, то ответ «`YES`», иначе «`NO`». Время работы решения — $\mathcal{O}(nm)$.

Задача F. Вежливость в метро

Автор задачи: Мария Жогова, разработчик: Константин Бац

Для простоты будем называть беременных женщин, пожилых людей и пассажиров с детьми особенными пассажирами. Заметим, что если все времена приходов особенных пассажиров ограничены сверху, то последний пришедший найдет себе сидячее место не позже, чем в $2 \cdot 10^5$. Действительно, так как особенных пассажиров не больше 10^5, всем им когда-то найдется место, а $a_i \leqslant 10^5$.

Сгруппируем всех обычных пассажиров по равным a_i. Для каждой такой группы предподсчитаем моменты t_j, когда пассажиры из этой группы будут отвлекаться от телефонов. По сказанному выше, нас интересуют $t_j \leqslant 2 \cdot 10^5$. Отсортируем все t_j по возрастанию и получим последовательность моментов, когда пассажиры из какой-то группы отвлекаются от телефонов.

По условию, если хотя бы один пассажир из группы встал, то все остальные тоже уступили свое место. Поэтому будем для каждой группы хранить флаг: встали пассажиры из этой группы, или нет.

Для решения задачи воспользуемся методом двух указателей: один указатель будет на первом особенном пассажире, который еще не вошел, а второй на моменте, когда обычные пассажиры из какой-то группы отрываются от телефонов. Из двух событий выберем то, которое произойдет раньше, или вход особого пассажира при равенстве времени.

Предположим, вошел особый пассажир. Проверим, если ли ранее освобожденное место: если есть, займем его, иначе добавим пассажира в очередь ожидающих свободного места.

Если же очередная группа обычных пассажиров отвлекается от телефонов, то:

- Проверим, что пассажиры из этой группы ране не освободили свои места.
- Если это так, то проверим, ожидает ли кто-то свободного места.
- Если кто-то ждет свободное место, освободим все места, занятые этой группой, а также группами, которые отрываются от телефонов в этот же момент.
- Посадим особых пассажиров на свободные места, если таковые появились.

Так как общее количество делителей всех целых чисел от 1 до k равно $\frac{k}{1} + \frac{k}{2} + \frac{k}{3} + \cdots + \frac{k}{k} = \mathcal{O}(k \log k)$, то на предподсчет всех t_j уйдет $\mathcal{O}(A \log A)$ времени, где $A = 2 \cdot 10^5$.

Тогда общее время работы решения — $\mathcal{O}\left(m + A \log A\right)$.

Задача G. Шоу фейерверков

Автор задачи и разработчик: Даниил Орешников

Упростим формулировку задачи: даны $n+1$ стек, вмещающие не более двух элементов, в них лежат n пар элементов с номерами от 1 до n; требуется перекладываниями верхних элементов из стеков добиться того, чтобы парные элементы лежали в одном стеке.

Заметим, что если в каждой паре один элемент лежит снизу какого-то стека, а другой — сверху, то можно справиться с задачей следующим образом. Выберем первый стек, переложим верхний элемент из него в «запасной» пустой. Пара к тому элементу, который остался внизу первого стека, лежит где-то наверху в другом — переложим его в первый. Продолжим то же самое для стека, из которого только что переложили верхний, и так далее, пока не найдется пара к самому первому отложенному элементу.

На упорядочивание каждого такого «цикла» размера k требуется $k+1$ действие, и всего есть не более $\frac{n}{2}$ таких циклов, поэтому хватит $\frac{3}{2}n$ действий.

Теперь обобщим это решение на случай, когда есть пары с двумя «верхними» или двумя «нижними» элементами (по их исходному расположению в стеках). Рассмотрим произвольный стек, пусть в нем лежат элементы 3 (сверху) и 4 (снизу). Посмотрим, лежит ли где-то 3 снизу — если да, то пусть, не теряя общности, верхний элемент в его стеке равен 2, продолжим то же рассуждение для 2, и аналогично продолжим для 4 в другую сторону. Рано или поздно, мы остановимся, либо замкнувшись в цикл, либо встретив с обеих сторон элемент, чья пара находится на той же глубине в другом стеке:

$$[x, 1$$
$$[2, 1$$
$$[3, 2$$
$$[4, 3$$
$$[5, 4$$
$$[5, y$$

В таком случае выложим два парных верхних элемента (1 и 1) в пустой стек, переместим 2, 3, 4 по циклу, как в предыдущем случае, дальше переложим y на x, и, наконец, переместим 5. В данном случае мы получили $k-1$ новых парных стеков, потратив $k+1$ действие, где k — длина такой цепочки. Поскольку цепочки имеют длину хотя бы 3, то если все стеки разбиваются на циклы или цепочки, нам хватит $2n$ действий даже с запасом.

Так же стоит аккуратно обрабатывать случай $x = y$, в котором цепочка зацикливается, и, в частности, случай двух одинаковых стеков, образующих две пары.

Для реализации этого алгоритма достаточно в явном виде хранить для каждого номера пары две позиции элементов из нее, а также сами стеки. При перемещениях можно просто изменять состояние стеком и обновлять информацию о том, где какой элемент находится. Так же следует поддерживать номер вспомогательного пустого стека, потому что он периодически меняется. Время работы решения — $\mathcal{O}(n)$.

Задача H. Новелла про осень

Автор задачи и разработчик: Даниил Орешников

Докажем следующий факт: если в новелле есть пара соседних букв c_1 и c_2, которые не являются соседними на клавиатуре, то ее нельзя напечатать, а иначе — можно. Действительно, после того, как будет напечатан символ c_1, палец писателя обязательно будет расположен на клавише с символом c_1. После чего, не печатая следующий символ, писатель может переместить палец только на другие клавиши с символом c_1. А для того, чтобы напечатать c_2, необходимо, чтобы клавиша с таким символом шла сразу после текущей.

В обратную сторону аналогично — если каждая пара соседних символов в новелле встречается подряд на клавиатуре, то чтобы напечатать очередной символ, достаточно переместить палец на первую клавишу соответствующей пары (c_1, c_2), и нажать следующую по кругу клавишу.

Таким образом, необходимо и достаточно проверить, что любая пара соседних символов в новелле встречается на клавиатуре на соседних позициях.

Для этого воспользуемся структурой данных «множество» (`unordered_set`, `HashSet`, `set`). Положим в множество все пары соседних символов на клавиатуре, после чего так же проитерируемся по всем парам соседних символов в новелле, и проверим принадлежность их пары множеству.

Время работы такого решения — $\mathcal{O}(n + |s|)$.

Задача I. Расстановка экспонатов

Автор задачи и разработчик: Даниил Орешников

Упорядочим все экспонаты по высоте, теперь $h_1 \leqslant h_2 \leqslant \ldots \leqslant h_n$. Заметим, что достаточно рассмотреть только $H = h_i$ для некоторого i, так как любое другое значение H можно опустить вниз до ближайшего h_i, не изменив множество, которое им ограничено.

Будем перебирать все возможные значения H по возрастанию от $h_{\frac{n}{2}}$ до h_n. Перебирать меньшие h_i тоже не имеет смысла, потому что тогда вне зависимости от W под критерий будет попадать строго меньше половины экспонатов.

Для очередного $H = h_i$, проверим, можно ли выбрать такое W, чтобы под описанный в условии критерий подходила ровно половина экспонатов. Для этого рассмотрим множество всех экспонатов с высотой $\leqslant H$, упорядочим их по высоте, и проверим, что ширина экспоната на позиции $\frac{n}{2}$ отличается от ширины экспоната на позиции $\frac{n}{2} + 1$.

Если они не отличаются, то не существует подходящего W. Действительно, мы не можем выбрать значение меньше ширины $\frac{n}{2}$-го экспоната, и не можем выбрать большее либо равное, потому что под такой критерий подходит уже хотя бы $\frac{n}{2} + 1$ экспонат. Если же эти две ширины оказались различными, то мы нашли еще один способ разбить экспонаты на два множества.

Однако, есть еще одна деталь, которую стоит учесть — не факт, что мы нашли действительно новый способ разбиения. Если при полученном W, равном $\frac{n}{2}$-й по величине ширине экспоната, ни один экспонат с высотой, равной H, не вошел в первую группу, то такой способ уже был посчитан ранее. Таким образом, необходимо так же проверить, что выбранный $W \geqslant \min\limits_{j\colon h_j = H}(w_j)$.

Осталось реализовать эту проверку за короткое время. Как и уже было сказано, будем рассматривать экспонаты по увеличению h_i. И будем поддерживать дерево поиска (например, декартово) по ширине на всех уже рассмотренных экспонатах. Когда встречаем следующее по величине значение h_i, добавляем в декартово дерево ширину каждого экспоната с такой высотой, после чего проверяем, что выполняется описанное выше свойство.

Время работы решения — $\mathcal{O}(n \log n)$.

Задача J. Уборка листьев

Автор задачи: Мария Жогова, разработчик: Владислав Власов

Отсортируем массив и посчитаем на нём префиксные суммы $p_i = a_1 + a_2 + \ldots + a_i$. Для этого достаточно сделать $p_0 = 0$ и посчитать в цикле каждый p_i как $p_{i-1} + a_i$. Так мы сможем узнавать сумму на любом отрезке массива a за $\mathcal{O}(1)$: $a_i + a_{i+1} + \ldots + a_j = p_j - p_{i-1}$.

Заметим, что достаточно перебрать $l = 1$ и $l = a_i + 1$ для всех возможных i. Действительно, если оптимален некоторый отрезок $[l, r]$ с другим l, его можно сдвинуть влево до ближайшего из указанных значений l, при чем слева в отрезок не войдут никакие новые a_i (потому что двигаемся до ближайшего $a_i + 1$), а справа точно ничего не добавится в отрезок, но возможно еще что-то даже выйдет за его рамки и сделает ответ лучше.

Таким образом, рассмотрим в качестве l единицу и все возможные $a_i + 1$, пока $r = a_i + k \leqslant c$, и выберем лучшую сумму. Находить для каждых l и r, какие a_i попадут в отрезок, можно либо двоичным поиском, либо методом двух указателей. Сумму на найденном отрезке можно будет найти с помощью префиксных сумм, посчитанных ранее.

Рассмотрим решение методом двух указателей: будем перебирать a_i в порядке возрастания для левой границы и искать последнее a_j, такое что $a_j \leqslant a_i + k$. Так как $a_i \leqslant a_{i+1}$, j для каждого следующего i будет не меньше, чем для $i - 1$, поэтому просто начнем перебирать j, начиная с найденного на предыдущей итерации значения. Поскольку j проходит по элементам массива, оно увеличится не больше n раз, и время работы такого решения будет $\mathcal{O}(n)$ плюс $\mathcal{O}(n \log n)$ на сортировку в начале.