Задача 1. Аделина и цветы

Автор задачи: Инесса Шуйкова, разработчик: Михаил Первеев

Подзадача 1

Для решения первой подзадачи переберем, сколько белых и красных роз попадет в букет. Пусть количество белых роз равно c_w ($0 \leqslant c_w \leqslant w$), а количество красных роз равно c_r ($0 \leqslant c_r \leqslant r$). Если $c_r \equiv 1 \pmod 2$ и $c_w + c_r \geqslant k$, у нас получится собрать букет с требуемыми свойствами. В противном случае, если ни для какой пары (c_w, c_r) данные условия не выполнены, собрать букет не удастся.

Мы получили решение, работающее за $\mathcal{O}(w \cdot r)$.

Подзадача 2

Во второй подзадаче будем перебирать, сколько красных роз будет взято в букет. Пусть это количество равно c_r ($0 \leqslant c_r \leqslant r$). Заметим, что если $c_r \equiv 1 \pmod 2$ и $c_r + w \geqslant k$, у нас получится собрать букет с требуемыми свойствами. В противном случае, если ни для какого значения c_r данные условия не выполнены, собрать букет не удастся.

Мы получили решение, работающее за $\mathcal{O}(r)$.

Подзадачи 3 и 4

Для решения данных подзадач достаточно заметить следующие факты.

Во-первых, если $r = 0$, собрать букет не удастся, так как мы не можем взять ни одной красной розы, а значит их количество всегда будет равно нулю. Ноль — четное число, поэтому такие букеты нам не подходят.

В противном случае будем собирать букет следующим образом: сначала возьмем максимально возможное количество красных роз, а затем все белые розы, и сравним суммарное количество роз с числом k.

Если r нечетно, то мы можем взять в букет r роз, соответственно, собрать букет удастся, если $r + w \geqslant k$.

Если r четно, то мы можем взять в букет лишь $r - 1$ роз, соответственно, собрать букет удастся, если $r + w - 1 \geqslant k$.

Также необходимо не забыть воспользоваться 64-битным типом данных, если вы реализуете решение на таких языках как C++, Java, Pascal или C#.

Мы получили решение, работающее за $\mathcal{O}(1)$.

Задача 2. Очередная игра на прямой

Авторы задачи: Кирилл Кудряшов и Михаил Кондрашин, разработчик: Михаил Кондрашин

Для начала выясним, как выглядит выигрышная стратегия игроков. Пусть Миша поставил свою фишку в клетку x. В какую клетку тогда выгоднее всего поставить свою фишку Егору? Пусть слева от фишки Миши находится l подарков, а справа — r. Тогда легко заметить, что Егор может получить как минимум $\max(l, r)$ подарков (надо встать в клетку $x - 1$ или $x + 1$). Заметьте, что Егору нет смысла вставать в другие клетки, так как тогда Миша может сначала начать двигаться в сторону Егора, а потом уже забрать все подарки с другой стороны.

Тогда в какую клетку нужно встать Мише? Если Егор всегда может набрать $\max(l, r)$ подарков, то Миша получает $n - \max(l, r)$. Поэтому нужно выбрать такую клетку, для которой значение $\max(l, r)$ минимально.

Подзадачи 1 и 2

В первых двух подзадачах достаточно было перебрать позицию Миши, а затем вычислить максимальное количество подарков, которые сможет собрать Егор, и вычесть это число из n. Очевидно, что Мише не выгодно ставить фишку левее, чем все подарки. Точно также ему не выгодно ставить фишку правее, чем все подарки. Поэтому существует $\mathcal{O}(\max(a_1, a_2, \ldots, a_n) - \min(a_1, a_2, \ldots, a_n))$ вариантов выбрать позицию. Для каждого из этих вариантов можно вычислить количество подарков слева и справа от позиции Миши за $\mathcal{O}(n)$. Таким образом, мы получили решение за $\mathcal{O}(n \cdot (\max(a_1, \ldots, a_n) - \min(a_1, \ldots, a_n)))$.

Подзадача 3

В данной подзадаче необходимо было заметить, что нет смысла вставать в клетки, в которых нет подарка. Ведь фишку, поставленную в клетку, в которой нет подарка, всегда можно переместить в клетку, в которой находится ближайший слева или ближайший справа подарок, и ответ от этого не уменьшится.

Тогда можно в качестве стартовой клетки перебирать только те клетки, в которых есть подарок. Таким образом мы получим решение за $\mathcal{O}(n^2)$.

Подзадачи 4, 5 и 6

Заметим, что $\max(l, r) \geqslant \left\lfloor \frac{n-1}{2} \right\rfloor$. Значит Миша точно не сможет получить больше подарков, чем $\left\lceil \frac{n}{2} \right\rceil$. Пусть подарки изначально отсортированы по возрастанию номера клетки, то есть $a_1 < a_2 < \ldots < a_n$. Тогда заметим, что на самом деле выгодно встать в подарок с номером $\left\lfloor \frac{n+1}{2} \right\rfloor$ (в 1-индексации). В этом случае мы всегда сможем забрать ровно $\left\lceil \frac{n}{2} \right\rceil$ подарков.

Таким образом, достаточно отсортировать массив, а затем вывести элемент с номером $\left\lfloor \frac{n+1}{2} \right\rfloor$ (в 1-индексации).

В зависимости от того, как будет реализована сортировка, решение получит баллы за те или иные подзадачи. В случае полного решения достаточно воспользоваться встроенной в язык программирования сортировкой. Время работы составит $\mathcal{O}(n \log n)$.

Задача 3. Бу, испугался, не бойся

Автор задачи: Александр Бабин, разработчик: Мария Жогова

Для начала заметим ключевую идею, которая будет использована во всех подзадачах: любые три прямые разных типов образуют равносторонний треугольник, кроме случая, когда эти три прямые пересекаются в одной точке. Таким образом, ответ на задачу равен $a \cdot b \cdot c - k$, где k — количество троек прямых разных типов, пересекающихся в одной точке.

Подзадача 1

В данной подзадаче достаточно перебрать все тройки прямых разных типов и проверить, что они пересекаются в одной точке. Пусть у нас есть прямая, параллельная оси OX, пересекающая ось OY в точке y_1; прямая, лежащая под углом $60°$ к оси OX, пересекающая ось OY в точке y_2; и прямая, лежащая под углом $120°$ к оси OX, пересекающая ось OY в точке y_3.

Составим уравнения данных прямых. Первая прямая будет иметь уравнение $y = y_1$, вторая прямая будет иметь уравнение $y = x\sqrt{3} + y_2$, а третья прямая — уравнение $y = -x\sqrt{3} + y_3$. Угловые коэффициенты второй и третьей прямых равны $\pm\sqrt{3}$, так как $\operatorname{tg} 60° = \sqrt{3}$ и $\operatorname{tg} 120° = -\sqrt{3}$.

Найдем x-координату точки пересечения второй и третьей прямых. Для этого необходимо решить уравнение $x\sqrt{3} + y_2 = -x\sqrt{3} + y_3$ относительно x. Получаем $x = \frac{y_3 - y_2}{2\sqrt{3}}$.

Теперь найдем y-координату точки пересечения второй и третьей прямых, подставив x-координату в уравнение второй прямой: $y = \frac{y_3 - y_2}{2\sqrt{3}} \cdot \sqrt{3} + y_2 = \frac{y_3 - y_2}{2} + y_2 = \frac{y_2 + y_3}{2}$.

Отсюда можно сделать вывод, что данные три прямые пересекаются в одной точке тогда и только тогда, когда $y_1 = \frac{y_2+y_3}{2}$.

Таким образом, мы получили решение за $\mathcal{O}(abc)$.

Подзадача 2

В данной подзадаче количество троек прямых, пересекающихся в одной точке, равно нулю. Это значит, что ответ равен $a \cdot b \cdot c$.

Таким образом, данная подзадача решается за $\mathcal{O}(1)$.

Подзадача 3

Научимся решать текущую подзадачу, воспользовавшись фактом из первой подзадачи. Переберем пару прямых, лежащих под углами 60° и 120° градусов к оси OX. Теперь мы знаем, что в одной точке с ними пересекаются только прямые, параллельные оси OX, имеющие y-координату $\frac{y_2+y_3}{2}$.

Так как в данной подзадаче никакие две прямые не совпадают, достаточно проверить, что существует прямая, параллельная оси OX, имеющая необходимую y-координату. Также в данной подзадаче y-координаты точек пересечения прямых с осью OY маленькие, поэтому мы можем воспользоваться этим. Создадим массив, в котором для каждой y-координаты сохраним значение 1, если существует прямая первого типа с такой y-координатой, и 0 в противном случае. Теперь во время перебора пар прямых, во-первых, проверим, что $y_2 + y_3$ четно, а во-вторых, проверим, что в массиве по индексу $\frac{y_2+y_3}{2}$ лежит значение 1.

Таким образом, мы решили подзадачу за $\mathcal{O}(bc)$.

Подзадача 4

В данной подзадаче необходимо сделать все то же, что и в предыдущей, но учесть случаи, когда прямые могут совпадать. Для этого в массиве для каждой y-координаты будем хранить не 1, если прямая первого типа с такой y-координатой есть, а количество прямых первого типа с такой y-координатой. Дальнейший ход решения аналогичен подзадаче 3.

Подзадача 5

Наконнец, для решения данной подзадачи можно заменить массив на структуру данных `std::unordered_map` в C++ или `dict` в Python для того, чтобы хранить количество прямых с данной y-координатой.

Полное решение задачи работает за $\mathcal{O}(bc)$.

Задача 4. Кирилл читает книги

Авторы задачи: Кирилл Кудряшов и Михаил Кондрашин, разработчик: Михаил Кондрашин

Подзадачи 1 и 2

Для решения первой и второй подзадач достаточно написать полный перебор. Можно перебрать подмножество дней, в которые Кирилл будет пить сок. Для каждого подмножества достаточно проверить, верно ли, что его размер не превосходит k. После этого для каждого подмножества необходимо вычислить количество прочитанных книг.

Время работы составит $\mathcal{O}(2^m \cdot (n+m))$.

Подзадача 3

Для решения третьей подзадачи необходимо перебирать только подмножества дней размера не более, чем k, так как у Кирилла есть всего лишь k банок сока. Для этого необходимо правильным образом организовать отсечения в переборе.

Время работы такого решения составит $\mathcal{O}(c \cdot (n+m))$, где $c = \binom{m}{0} + \binom{m}{1} + \ldots + \binom{m}{k}$.

Подзадача 4

В данной подзадаче у Кирилла нет банок с соком. Потому для решения достаточно реализовать наивный алгоритм решения — каждый день Кирилл будет читать максимальное количество книг, которые он сможет прочесть за данный день. Будем поддерживать указатель на последнюю непрочитанную книгу, и в каждый день двигать его вправо до тех пор, пока у Кирилла не закончится дневной лимит на количество прочтенных страниц, либо пока не закончатся книги.

Время работы такого решения составит $\mathcal{O}(n+m)$.

Подзадача 5

В данной подзадаче каждая книга состоит из одной страницы. Это значит, что Кириллу не важно в какие дни он будет пить сок, так как от этого количество прочитанных книг не изменится. Поэтому, например, можно выпить сок в первые $\min(m, k)$ дней, после чего реализовать алгоритм из подзадачи 4 для подсчета количества прочтенных книг.

Время работы такого решения составит $\mathcal{O}(n+m)$.

Подзадача 6

В данной подзадаче у Кирилла есть всего лишь одна банка сока. Переберем день, в который Кирилл выпьет эту банку сока. После этого запустим решение из подзадачи 4, которое вычислит количество прочитанных книг. Из полученных ответов выберем наибольший.

Данное решение будет работать за $\mathcal{O}(m \cdot (n+m))$.

Подзадача 7

Для решения задачи на полный балл воспользуемся методом динамического программирования. Пусть $dp[i][j]$ — максимальное количество книг, которые сможет прочитать Кирилл, если прошло m дней, и он выпил j банок сока.

Базу динамики можно описать следующим образом: $dp[0][0] = 0$ (прошло 0 дней, Кирилл выпил 0 банок сока и прочитал 0 книг).

Рассмотрим, как вычисляется значение $dp[i][j]$. Введем вспомогательную функцию $f(p, c)$, равную количеству книг, которые сможет прочитать Кирилл, если он начнет чтение с книги под номером p и может прочитать не более, чем c страниц. О том, как вычислять значение данной функции, мы поговорим позже.

1. Если Кирилл не будет пить сок в i-й день, тогда $dp[i][j] = dp[i-1][j] + f(dp[i-1][j], b[i])$.
2. Если Кирилл будет пить сок в i-й день, тогда $dp[i][j] = dp[i-1][j-1] + f(dp[i-1][j-1], b[i]+x)$.

В качестве значения $dp[i][j]$ нужно выбрать максимальное из данных двух значений. Теперь обсудим, как вычислить значение функции $f(p, c)$.

Для вычисления значения данной функции можно воспользоваться двоичным поиском. Будем перебирать количество прочитанных книг при помощи двоичного поиска. Пусть мы хотим проверить, можно ли прочитать t книг. Для этого достаточно проверить, что $a_p + a_{p+1} + \ldots + a_{p+t-1} \leqslant c$.

Для того, чтобы вычислить данную сумму за $\mathcal{O}(1)$, можно предподсчитать массив префиксных сумм над массивом a.

Таким образом, для вычисления каждого состояния динамики нам нужно выполнить двоичный поиск, который будет работать за $\mathcal{O}(\log n)$. Всего в динамике $\mathcal{O}(mk)$ состояний.

Таким образом, мы получили решение, которое работает за $\mathcal{O}(mk \log n)$.

Задача 5. Хорошие раскраски — 2024

Автор задачи: Александр Бабин, разработчик: Михаил Первеев

Подзадача 1

Для решения первой подзадачи можно было воспользоваться методом полного перебора. Так как $k = 3$, а количество спичек равно $3n + 1$, количество всевозможных раскрасок равно 3^{3n+1}. Далее каждую такую раскраску необходимо проверить и выяснить, является ли она хорошей. Эту проверку можно выполнить за $\mathcal{O}(n)$.

Для того, чтобы реализация перебора была проще, можно сформулировать задачу в терминах графов. Построим неориентированный граф, в котором каждой спичке будет соответствовать вершина. Соединим ребрами все пары спичек, которые являются соседними. Теперь необходимо вычислить количество способов раскрасить вершины графа в три цвета таким образом, чтобы никакие две вершины, соединенные ребром, не были окрашены в один цвет.

Таким образом, мы получили решение за $\mathcal{O}(3^{3n} \cdot n)$.

Подзадача 2

Данная подзадача отличается от предыдущей лишь ограничением на n. Заметим, что в наше переборное решение предыдущей подзадачи можно добавить отсечения, если реализовать перебор в виде рекурсии. Если в какой-то момент после покраски очередной вершины оказалось, что она соединена ребром с некоторой вершиной такого же цвета, продолжать перебор не имеет смысла. Так как ответ при максимальном для данной подзадачи значении $n = 20$ равен 6 291 462, данное решение будет работать достаточно быстро.

Подзадача 3

Данная подзадача отличается от двух предыдущих ограничением на n. Заметим, что в данной подзадаче всего 10 тестов для всесозможных значений n от 21 до 30. Таким образом, мы можем локально предподсчитать ответы на все тесты данной подзадачи. Запустим решение, полученное в подзадаче 2, для всех n от 21 до 30 и сохраним ответы в массив.

При $n = 30$ переборное решение с отсечениями будет работать на среднем компьютере от 5 до 15 минут.

Подзадача 4

В данной подзадаче $n = 1$, однако k может быть большим. Если снова сформулировать задачу в терминах графа, как это было сделано в решении подзадачи 1, мы получим следующую формулировку. Необходимо вычислить количество способов раскрасить вершины цикла из четырех вершин в k цветов таким образом, чтобы никакие две соседние вершины не были окрашены в один цвет.

Рассмотрим два случая.

1. **Верхняя и нижняя спички окрашены в разные цвета.** Тогда у нас есть k способов зафиксировать цвет верхней спички и $k - 1$ способов зафиксировать цвет нижней спички. Каждая из боковых спичек должна иметь цвет, отличный от верхней и нижней спичек, поэтому

каждую из боковых спичек можно окрасить в $k-2$ различных цвета. Таким образом, итоговое количество способов равно $k \cdot (k-1) \cdot (k-2)^2$.

2. **Верхняя и нижняя спички окрашены в один цвет.** Тогда у нас есть k способов зафиксировать этот цвет. Каждая из боковых спичек должна быть окрашена в другой цвет, поэтому для каждой из них есть $k-1$ способ выбрать цвет. Таким образом, итоговое количество способов равно $k \cdot (k-1)^2$.

Итоговое количество раскрасок равно $k \cdot (k-1) \cdot (k-2)^2 + k \cdot (k-1)^2$.
Таким образом, мы получили решение подзадачи за $\mathcal{O}(1)$.

Подзадача 5

В данной подзадаче были маленькие ограничения на n и k, позволяющие воспользоваться методом динамического программирования. Пусть $dp[l][c_1][c_2][c_3]$ — количество различных хороших раскрасок l квадратов при условии, что в последнем квадрате верхняя спичка имеет цвет c_1, правая спичка имеет цвет c_2, а нижняя спичка — цвет c_3.

База динамики — значения $dp[1][c_1][c_2][c_3]$, которые вычисляются тривиально. Данное значение равно нулю, если $c_1 = c_2$ или $c_2 = c_3$, $k-1$, если $c_1 = c_3$, либо $k-2$, если $c_1 \neq c_3$.

Для того, чтобы выполнить переход от полоске из l квадратов к полоске из $l+1$ квадратов достаточно перебрать цвета новых трех спичек и проверить, что никакие две соседние спички не окрашены в одинаковый цвет.

Данная динамика имеет $\mathcal{O}(n \cdot k^3)$ состояний. Из каждого состояния есть $\mathcal{O}(k^3)$ переходов. Таким образом, мы получили решение за $\mathcal{O}(n \cdot k^6)$.

Подзадача 6

Данная подзадача является ключевой при решении всех последующих задач. Заметим, что при решении предыдущей подзадачи можно было уменьшить количество состояний динамики следующим образом. Для последнего квадрата нам важно лишь, равны ли цвета верхней и нижней спичек.

Поэтому будем считать новую динамику следующим образом. Пусть $dp[l][0]$ — количество хороших раскрасок l квадратов при условии, что верхняя и нижняя спички последнего квадрата окрашены в разные цвета. Аналогично, пусть $dp[l][1]$ — количество хороших раскрасок l квадратов при условии, что верхняя и нижняя спички последнего квадрата окрашены в один цвет.

База динамики — значения $dp[1][0]$ и $dp[1][1]$, которые были вычислены при решении подзадачи 4.

Для пересчета динамики необходимо рассмотреть четыре различных случая.

1. **В старом квадрате верхняя и нижняя спички окрашены в разные цвета, в новом квадрате верхняя и нижняя спички окрашены в разные цвета.** Вычислим количество способов выбрать цвета для трех добавленных спичек. Верхняя спичка может быть окрашена в любой из $k-2$ цветов. Нижняя спичка, аналогично, может быть окрашена в любой из $k-2$ цветов. Наконец, правая спичка также может быть окрашена в любой из $k-2$ цветов, так как цвета верхней и нижней спичек различны. Таким образом, получаем $(k-2)^3$ способов. Однако, среди этих способов могут оказаться варианты, когда верхняя и нижняя спички окрашены в один цвет. Поэтому из данного количества необходимо вычесть такие плохие случаи. Если верхняя и нижняя спички окрашены в один цвет, они могут принимать любой из $k-3$ цветов, так как они не могут совпадать с предыдущими тремя спичками. Таким образом, необходимо вычесть из нашего ответа величину $(k-2) \cdot (k-3)$. Итоговое количество вариантов равно $(k-2)^3 - (k-2) \cdot (k-3)$.

2. **В старом квадрате верхняя и нижняя спички окрашены в один цвет, в новом квадрате верхняя и нижняя спички окрашены в разные цвета.** Вычислим количество способов выбрать цвета для трех добавленных спичек. Верхняя спичка может быть окрашена в

$k-2$ цветов, а нижняя спичка — в $k-3$ цветов, чтобы ее цвет отличался от верхней спички. Наконец, правая спичка может быть окрашена в $k-2$ цветов. Итоговое количество вариантов равно $(k-2)^2 \cdot (k-3)$.

3. **В старом квадрате верхняя и нижняя спички окрашены в разные цвета, в новом квадрате верхняя и нижняя спички окрашены в один цвет.** Вычислим количество способов выбрать цвета для трех добавленных спичек. Верхняя (а значит и нижняя) спички могут быть окрашены в любой из $k-3$ цветов. Правая спичка может быть окрашена в $k-1$ цвет. Итоговое количество вариантов равно $(k-1) \cdot (k-3)$.

4. **В старом квадрате верхняя и нижняя спички окрашены в один цвет, в новом квадрате верхняя и нижняя спички окрашены в один цвет.** Вычислим количество способов выбрать цвета для трех добавленных спичек. Верхняя (а значит и нижняя) спички могут быть окрашены в любой из $k-2$ цветов. Правая спичка может быть окрашена в $k-1$ цвет. Итоговое количество вариантов равно $(k-1) \cdot (k-2)$.

Данные коэффициенты позволяют нам переходить от значений $(dp[l][0], dp[l][1])$ к значениям $(dp[l+1][0], dp[l+1][1])$, а значит мы можем вычислить все необходимые значения динамики.

Таким образом, мы получили решение за $\mathcal{O}(n)$.

Подзадача 7

Воспользуемся динамикой из подзадачи 6 и немного модифицируем ее. Будем хранить состояние не только для последнего квадрата, но и для первого. Таким образом, теперь мы будем вычислять значение динамики $dp[l][s][t]$, где $s, t \in \{0, 1\}$, параметр s отвечает за равенство цветов верхней и нижней спичек первого квадрата, а параметр t отвечает за равенство цветов верхней и нижней спичек последнего квадрата. Данная динамика может вычисляться аналогично динамике из предыдущей подзадачи. Нужно лишь правильно инициализировать значения $dp[1][s][t]$.

Теперь воспользуемся идеей корневой оптимизации. Зафиксируем некоторое число B. Вычислим значение нашей динамики для всех $l \leqslant B$. Теперь заметим, что последовательность из n квадратов можно разбить на не более, чем $\frac{n}{B}$ блоков длины B, а также, возможно, один блок длины меньше B. Теперь заметим, что для каждого из блоков у нас уже посчитано значение динамики. Единственное, что осталось сделать — объединить значения динамики для всех блоков, чтобы получить из них ответ для n квадратов.

Пусть есть два блока с длинами l_1 и l_2, для которых вычислены значения динамики $dp[l_1][s][t]$ и $dp[l_2][s][t]$ для всех $s, t \in \{0, 1\}$. Заметим, что из данных двух значений можно легко получить значение динамики $dp[l_1 + l_2][s][t]$ для всех $s, t \in \{0, 1\}$. Для этого необходимо перебрать состояние последнего квадрата первого блока и первого квадрата второго блока, а затем рассмотреть четыре случая, описанных в решении подзадачи 6. Таким образом, мы научились объединять ответы для двух динамик за $\mathcal{O}(1)$.

Теперь необходимо воспользоваться данной техникой объединения и объединить ответы для всех имеющихся блоков. Первая часть решения, вычисляющая значение динамики для всех $l \leqslant B$, работает за $\mathcal{O}(B)$. Вторая часть решения, объединяющая значения динамики, работает за $\mathcal{O}\left(\frac{n}{B}\right)$. Таким образом, решение работает за $\mathcal{O}\left(B + \frac{n}{B}\right)$, что при $B = \sqrt{n}$ равно $\mathcal{O}(\sqrt{n})$.

Подзадача 8

Для решения данной подзадачи необходимо воспользоваться идеей из предыдущей подзадачи и оптимизировать ее. Рассмотрим запись числа n в двоичной системе — пусть $n = 2^{\alpha_0} + 2^{\alpha_1} + \ldots + 2^{\alpha_k}$, где $\alpha_i \neq \alpha_j$ при $i \neq j$, и $\alpha_i \leqslant \log_2 n$ для всех i.

Идея решения будет заключаться в том, что сначала мы вычислим значения динамики для всех длин l, являющихся степенями двойки, не превосходящими $\log_2 n$. Иными словами, мы хотим вычислить значения $dp[1][s][t], dp[2][s][t], dp[4][s][t], \ldots, dp[2^{\lfloor \log_2 n \rfloor}][s][t]$ для всех пар $s, t \in \{0, 1\}$. После

этого можно будет применить технику объединения из подзадачи 7 и, объединив значения динамики $dp[2^{\alpha_0}][s][t], dp[2^{\alpha_1}][s][t], \ldots, dp[2^{\alpha_k}][s][t]$, найти ответ. Данная часть решения будет работать за $\mathcal{O}(\log n)$, если нам известны значения динамики для всех степеней двойки.

Наконец, научимся вычислять значения динамики для всех степеней двойки. Значение $dp[1][s][t]$ можно легко вычислить, так как это мы уже неоднократно делали ранее, когда рассматривали базу динамики. Теперь заметим, что $2^k = 2^{k-1} + 2^{k-1}$, и для вычисления значения $dp[2^k][s][t]$ достаточно воспользоваться техникой объединения для значений $dp[2^{k-1}][s][t]$ и $dp[2^{k-1}][s][t]$ (то есть, мы объединяем динамику саму с собой). Таким образом, данная часть решения также будет работать за $\mathcal{O}(\log n)$, так как нам придется сделать $\mathcal{O}(\log n)$ объединений динамики.

Итого, получилось решение, работающее за $\mathcal{O}(\log n)$.