Разбор задачи «Фасады»

В маленьких ограничениях достаточно просто дважды — начиная красить первый фасад в *первый цвет*, а затем начиная красить первый фасад во *второй цвет*, выполнить симуляцию процесса покраски фасадов, последовательно стартуя с каждого дома и выбирая наилучший результат.

Чтобы решить задачу на полный балл, нужно действовать следующим образом. Предположим, что первые $\#j$ фасадов выкрашены в *первый цвет*, а фасады домов, начиная с $\#(j+1)$ — во *второй цвет*. При этом нам известно суммарное количество неизрасходованной краски *первого цвета* $sa[j]$ и суммарное количество неизрасходованной краски *второго цвета* $sb[j]$. Если теперь мы подвинем границу цветов вправо, то получить $sa[j+1]$ и $sb[j+1]$ можно несложным пересчетом: $sa[j+1] = sa[j] + ra[j]$, $sb[j+1] = sb[j] - rb[j]$, где $ra[j] = (a - s[j] mod a) mod a$, а $rb[j] = (b - s[j] mod b) mod b$ —количество краски первого и второго цвета соответственно, которая останется неизрасходованной при покраске фасада $\#j$.

Следует рассмотреть два случая: когда сначала фасады красят в первый цвет, а потом во второй, и когда сначала фасады красят во второй цвет, а потом в первый. Для каждого случая можно найти номер фасада $\#k$, обеспечивающий наилучший результат (наименьшее суммарное количество неизрасходованной краски), а среди наилучших запоминать именно тот $\#k$, который ближе к центру. В итоге останется выбрать из двух наилучших результатов и вывести ответ.

Все действия, описанные в задаче, имеют линейную асимптотику.

Разбор задачи «И о погоде»

Поскольку номера сортов травы упорядочены по приемлемым для них температурам, то, если температура приемлема для сорта $\#j$, то она приемлема для всех сортов, имеющих меньшие номера.

Проведем предварительную подготовку. Для каждого из m дней вычислим наибольший номер сорта травы, которую можно в этот день посадить. Очевидно, что этот номер для дня $\#i$ определяется минимальным значением t на отрезке от дня $\#i$ до дня $\#m$. Можно, проходя от конца к началу массива $t[]$, сформировать массив $toGrass[]$, элементами которого и будут являться эти номера.

Кроме того, преобразуем план. Обратим внимание, что если в плане $s[j+1] < s[j]$, то это означает, что более морозоустойчивая трава должна быть посажена позже менее морозоустойчивой. Однако поскольку менее морозоустойчивая трава не должна погибнуть до истечения дня $\#m$, то все дни после ее посадки будут для нее приемлемы по температуре. Поэтому мы можем заменить в плане величину $s[j+1]$ на $s[j]$. Пройдем от начала до конца плана, проводя такие замены. В результате получим план, записанный в виде неубывающей последовательности номеров.

Теперь найдем, в какой день мы могли бы посадить траву первого сорта, предписываемого планом. Пусть это будет день $\#k$. Далее заведем счетчик выполненных элементов плана и будем двигаться по массиву $toGrass[]$ вправо. Каждый раз, когда очередной элемент этого массива будет позволять выполнить очередной элемент плана, будем увеличивать счетчик на 1. Заметим, что в преобразованном плане могут идти подряд одинаковые элементы, однако, поскольку это в действительности разные сорта трав, нам нужно, чтобы каждому такому элементу в массиве $toGrass[]$ соответствовал хотя бы один день.

Когда план будет исчерпан, проанализируем номер позиции $\#pos$, на которой мы оказались в массиве $toGrass[]$. Если она отстоит от нашей стартовой позиции $\#k$ ровно на d дней, мы получили ответ. Если же это не так, потребуется предпринять еще некоторые шаги. Если мы ушли от $pos - k >= d$, то ответом будет день $\#(pos - d + 1)$. А вот если день $\#pos$ находится слишком близко к дню $\#k$, то нам нужно будет попытаться подвинуться вправо на недостающее количество дней — разумеется, если это возможно в принципе. Если это невозможно, или же план не был исчерпан, а мы уже дошли до дня $\#m$ — то ответа не существует, и следует вывести -1.

Разбор задачи «Монтаж»

Бригада не обязана каждый день устанавливать максимально возможное количество кресел. Однако чем ниже будет эта «планка», тем в более выгодном свете бригада предстанет перед самым главным инспектором.

Эту задачу можно решить методом бинарного поиска по ответу. Поясним, почему этот метод может быть применен. Если у нас есть некоторое число k, такое, что при данном максимально

возможном количестве бригада успевает установить все зафиксированные инспекторами кресла в сроки, которые не противоречат сведениям о доставке и о проведении инспекций, то любое большее число также является подходящим. В точности так же, если у нас есть некоторое число $\#k$, которое не позволяет вписать работу бригады в имеющийся график, то это будет верно и для всех чисел, не превосходящих его.

Остается только научиться достаточно быстро проверять, можно ли установить все кресла для некоторого числа $\#k$, соблюдая график. Это несложно делается «жадным» образом. Нам всегда доступна информация о том, какое количество кресел бригада должна установить в промежутке от одного визита инспектора до другого. Кроме того, мы можем поддерживать количество кресел, которые уже привезены, но еще не установлены. В каждый день от одного визита до другого будем устанавливать максимально возможное количество уже привезенных кресел (разумеется, устанавливать лишние нельзя).

Асимптотика решения $O(clogc)$, где c — общее количество кресел.

Разбор задачи «Оптом дешевле»

Для решения подзадачи 1 достаточно обратить внимание, что возможных ответов не более 2^n, поэтому достаточно перебрать все 2^n вариантов и для каждого посчитать итоговую стоимость за n. С использованием битовых масок или рекурсивного перебора сложность решения $O(2^n \cdot n)$.

Для решения основной подзадачи воспользуемся принципом динамического программирования. Любой ответ представим в виде чередующихся отрезков из букв 'A' и 'B'. Соответственно, нашим состоянием можно сделать двойку (индекс, последняя буква).

Но ведь по условию цена одного участка зависит не только от буквы, но и еще от количества подряд идущих букв. Это можно решить следующим образом: для подсчета значения в текущем индексе мы переберем, где начался последний отрезок с заданной буквой. Тогда рекуррентное соотношение примет следующий вид:

$dp[index][last] = min(dp[k][other] + price(last, i-k) \cdot (i-k))$,

где - $other$ - "другая"буква относительно буквы "last"(для 'A' это 'B' и наоборот); - $price(type, length)$ - цена посадки одного участка травой типа $type$ на отрезке длиной length (розничная или оптовая); - k принимает все значения от $start[index][last]$ до $i-1$ включительно. Но чему же равно $start[index][last]$?

Вспомним, что k в нашей формуле - это конец отрезка, состоящего из букв $other$, а все буквы на отрезке $k+1 \dots i$ равны букве $last$. Но если мы допустим, что какая-то из букв на этом отрезке фиксирована и равна букве $other$, то все наши вычисления становятся некорректными - мы рассматриваем ответ, который просто невозможно получить! Соответственно, становится ясно, что $start[index][last] = max(j | j \leqslant i$ and $s[j] == other)$ - наибольшая позиция, на которой встретилась буква $other$ - мы не можем продлить отрезок дальше, так как этим мы нарушим изначальную фиксацию.

Для решения подзадачи 2 было достаточно реализовать вычисление этой рекуррентной формулы с поиском минимума среди подходящих индексов за линейное время - итоговая сложность решения получается $O(n^2)$.

Для подзадачи 3 требуется оптимизировать поиск индекса k, минимизирующего правую часть выражения в рекурренте.

Во-первых, давайте заметим, что $price(last, i-k)$ принимает всего два возможных значения: если $i-k < d[last]$, то цена равна $p[last]$, а иначе - $s[last]$. Если мы преобразуем данное неравенство, то получим разбиение формулы на 2:

- $dp[index][last] = min(dp[index][last], dp[k][other] + s[last] \cdot (i-k))$, $start[index][last] \leqslant k \leqslant i-d$
- $dp[index][last] = min(dp[index][last], dp[k][other] + p[last] \cdot (i-k))$, $i-d < k < i$

То есть мы перешли к двум независимым отрезкам, на каждом из которых ищем минимум для соответствующей формулы.

Но что нам дало такое разбиение? Рассмотрим отрезок, соответствующий s[last] (другой рассматривается аналогично): мы ищем минимум выражения

$min(dp[k][other] + s[last] \cdot (i-k)) = min((dp[k][other] - s[last] \cdot k) + s[last] \cdot i) = min(F[k][other] + s[last] \cdot i) = min(F[k][other]) + s[last] \cdot i$,

то есть мы перешли к поиску значения выражения, зависящего полностью от индекса k и типа буквы $other$, но никак не зависящего от текущего индекса i. Это является классической задачей поиска минимума на отрезке с изменением значений по индексу, что может быть выполнено эффективно с помощью различных структур данных. Например, используя дерево отрезков, мы получаем $O(logN)$ на запрос, что дает итоговую сложность решения $O(NlogN)$.