Бабушкин А.

1 Обязательные задачи

1.
a) `dp[i] =` $\sum_{j=max(0,i-k)}^{i-1}$`dp[j], dp[0] = 1`
b) В предыдущем пункте всегда надо суммировать последние k элементов динамики → держим сумму на очереди.
c) Воспользуемся тем же приёмом, что и при подсчёте чисел Фибоначчи – возведём матрицу k x k в степень n.

2.
a) Заведём массив `dp` размера S + 1. `dp[i] =` максимальная цена, которую можно набрать с весом `i`. Изначально `dp[0] = 0, i` $\neq$ `0` $\rightarrow$ `dp[i] = -INF`. Затем обрабатываем предметы по одному, проходимся по всем `i` $\in$ `[0..S]` по возрастанию, релаксируем `dp[i +` w_j`] max= dp[i +` w_j`], dp[i] +` v_j.
b) То же самое, что и в предыдущем пункте, только $\forall j$ $w_j = v_j$, плюс храним не только `dp`, но и какой последний предмет мы использовали, чтобы прийти в вес i. Тогда восстановление ответа – это просто откатиться по таким "предкам" от S до 0.
c) То же, что и пункт b, но надо вернуть цены, и проходиться по весам в порядке убывания, а не возрастания. Так мы каждый предмет используем не больше одного раза. Восстановление ответа будем начинать не с S, а с максимального `dp`.

3. Для каждого элемента первого массива найдём и запишем в третий массив позицию его во втором массиве (или просто будем его игнорировать, если такого же во втором массиве нет). Это можно сделать за O(n) хэш-таблицей. Затем найдём в третьем массиве НВП. Понятно, что она соответствует какой-то подпоследовательности обоих массивов, причём общей.

4. Для каждого элемента будем считать два ответа – максимальная пилообразная последовательность, которая заканчивается в нём, при этом он больше предпоследнего элемента, или он меньше. Как пересчитывать? Разберём случай, когда наш элемент должен оказаться меньше предпоследнего. Тогда надо пробежаться по всем элементам слева от него и больших его, и обновить ответ с помощью их противоположного значения (то есть они должны были быть больше своего предпоследнего, т.к. должна быть пилообразность). Это решение за квадрат. Но если вместо этого делать запрос на максимум на суффиксе дерева отрезков (ДО по значениям элементов, а не индексам), то будет O($n \log n$).

5. `dp[i][j] =` максимальный общий префикс суффиксов i и j, нумерация с нуля, то есть суффикс n – пустой. Тогда $\forall i$ `dp[i][n] = 0, dp[i][j] = dp[i + 1][j + 1] + 1`, если `s[i] == s[j]`, иначе `dp[i][j] = 0`.

6. `dp[i] =` минимальное число символов, чтобы набрать префикс i. Всегда можно

обновить dp[i] = dp[i - 1] + 1, но нам интересны обновления с помощью скобочек. Пусть последним действием на префиксе cur была скобочка (n,i), то есть мы скопировали n символов с i-ой позиции. Переберём n. Тогда подстрока `s[cur-n+1..cur]` уже где-то встречалась. Если нет, то с помощью такого n обновиться нельзя. Если да, то нам выгодно взять самое левое её вхождение и обновить `dp[cur]` с помощью `dp[cur-n]` и длы записи (n,i), т.к. если мы начинаем копировать в позицию `cur-n+1`, то до `cur-n` всё должно быть набрано. Как знать длину записи (n,i)? Для каждого числа до n предподсчитать его длину в десятичной записи. Как знать самое левое вхождение строки? Посчитать хэши всех подстрок и для каждого хэша его самое левое вхождение. Итого решение за квадрат времени и памяти.

2 Дополнительные задачи

1. Перебираем предметы. В каждом предмете бежим по весам в порядке возрастания. Чтобы обновить текущее значение, мы можем использовать `dp[i - w], dp[i - 2w], ..., dp[i - kw]`. Тогда если перебирать все i с одним значением по модулю w, то это, казалось бы, максимум на очереди. Однако если мы обновляемся с помощью `dp[i - xw]`, то мы используем значение `dp[i - xw] + xv`, то есть вес предмета зависит от его положения в очереди. Это можно обработать так: пусть мы рассматриваем предмет под номером i. Тогда в очередь мы положим не `dp[i]`, а `dp[i] - (i / w)v`. Когда же мы вытащим максимум max из очереди, `dp[i]` мы обновим не с помощью max, а с помощью `max + (i / w)v = (dp[j] - (j / w)v) + (i / w)v = dp[j] + v(i - j)/w`, то есть обновление будет корректное.

2. b) `dp[n][k]` = кол-во перестановок из n элементов с k минимумами. `dp[1][0] = 1`. Как пересчитывать? Будем обновлять вперёд. Пусть у нас есть перестановка на n элементах. Представим, что они пронумерованы не [1..n], а [2..n+1]. Тогда нам нужно добавить единицу, чтобы получить перестановку на n + 1 элементе. Есть n + 1 вариант, куда поставить эту единицу. Если она окажется с краю, то не будет локальным минимумом. Если окажется рядом с бывшим минимумом, то будет им сама, но прошлый локальный минимум перестанет таким быть. То есть в этих случаях кол-во минимумом не изменится. Если же поставить её между двумя не-минимумами, то она станет новым минимумом и их число увеличится. Сколько есть мест рядом с локальными минимумами, если минимумов k? Очевидно, 2k, причём все эти места разные, т.к. два минимума рядом не стоят. Тогда пересчёт из `dp[n][k]` такой:
   `dp[n + 1][k] += 2(k + 1)dp[n][k],`
   `dp[n + 1][k + 1] += (n + 1 - 2(k + 1))dp[n][k].`

3.

4.