1 Обязательная часть

1. ***Рюкзак с ценами.***
   Решается, как задача о рюкзаке (без стоимостей), но теперь в массиве храним не 1/0 (можно получить/нельзя), а максимальная стоимость, которой добьемся, взяв первые k предметов и рюкзак вместимости s. То есть $d[k][s] = max(d[k-1][s], d[k-1][s-w_k] + p_k)$. В ячейке $d[n][w]$ будет лежать ответ.

2. ***Быстрый НОП.***
   Мы умеем находить наибольшую возрастающую последовательность за $\mathcal{O}(n \log n)$. Сведем НОП к этой задаче. Пусть у нас есть массивы a и b. Сделаем их копии a', b' и отсортируем их. Это все делается за $\mathcal{O}(n \log n)$. Теперь будем бежать по a, для каждого элемента за логарифм при помощи бинпоиска узнавать, есть ли он в b' и запоминать в массив a", если есть. То же самое сделаем для b и получим b". Теперь у нас есть два массива с одинаковым набором элементов. Бежим по b", каждый элемент записываем в новый массив пар c - пара <элемент, позиция в b">. Отсортируем c. Бежим по a" и при помощи бинпоиска ищем каждый элемент в массиве c, в массив d записываем позицию этого элемента (второе значение найденной пары). Найдем наибольшую возрастающую подпоследовательность d. Найденные индексы - индексы элементов НОП исхожных массивов в массиве b".

   Работает алгоритм за $\mathcal{O}(n \log n)$, так как ничего дольше бинпоиска для всех элементов в массиве мы не делаем. Почему работает. Проще с примером:

   $$a = 5,3,2,1,4,6 \quad b = 7,6,9,1,3,2$$
   $$a' = 1,2,3,4,5,6 \quad b' = 1,2,3,6,7,9$$
   $$a'' = 3,2,1,6 \quad b'' = 6,1,3,2$$

   Этими действиями мы просто убрали из изначальных массивов все эл-ты, встречающиеся лишь в одном массиве. $c = <1,2>, <2,4>, <3,3>, <6,1>$
   Это массив нужен, чтобы искать позиции эл-тов массива a не за квадрат, а за $n \log n$.
   $d = 3,4,2,1$
   Теперь мы получили соответствие: iый элемент массива a" стоит на d[i] позиции в массиве b". Тогда очевидно, что их наибольшая общая подпоследовательность - наибольшая возрастающая (это определяет правило, что элементы стоят строго друг за другом, а не где попало) подпоследовательность в d. Ответ получится:
   позиции 3, 4, что соответствует элментам 3, 2.

3. ***Наибольшая пилообразная подпоследовательность.***
   Динамика. Заводим массив f, где f[i] — максимальная длина последовательности, оканчивающейся на элементе с индексом i, такая что последние два элемента в ней расположены в порядке убывания. Заводим g, где g[i] - максимальная длина последовательности, оканчивающейся на элементе с индексом i, такая что последние два элемента в ней расположены в порядке возрастания.

   $$f[i] = 1 + maxg[j], \text{ среди всех } j < i \text{ таких, что } a[j] > a[i]$$
   $$g[i] = 1 + maxf[j], \text{ среди всех } j < i \text{ таких, что } a[j] < a[i]$$

4. ***LZSS***

Для каждого префикса посчитаем z-функцию для него перевернутого ($s[i]s[i-1]...s[1]$). Это делается за n для каждого элемента, то есть времени уйдет $\mathcal{O}(n^2)$. Теперь, как восстановить ответ. "Конец"каждого ответа - либо буква, либо скобока (n, j). То есть ans[i] либо $ans[i-1]+s[i]$ (ставим букву), либо $ans[j](i-j,k)$.

Как нам найти k. Для префикса 0..i мы посчитали z-функцию и запомнили, в каком индексе начинается самая длинная подстрока, равная префиксу строки s[i]s[i-1]...s[1] и само это максимальное значение, которое достигается. Если это значение равно m и верно, что $m > i-j$ и достигается в индексе l, то возможный новый ответ для $ans[i]\ \ -ans[j]+(i-j,l)$.

Но считать z-функцию мы будем ее не по-простому, а для каждого суффикса сделаем "грязный хак": считаем для префикса $0..n-1$. Если подстрока длинны l, которая матчится с суффиксом этого префик- са (n-l..n-1), то для префиксов $n-l, n-l+1, ..., n-1$ мы говорим, что уже нашли для них хорошую строку. Это все делается потому, что ans[j] мог содержать на конце скобочки (x, y) или (x, y1), при этом, если бы мы начали копировать из y, то получили бы префикс 0..i, а если из y1, то нет. Тогда нужно как-то понимать, что ответ для префикса j должен заканчиваться именно скобочкой (x, y), а не (x, y1). Такой подсчет z-функции решает данную проблему. Теперь для меньшего и большего префикса мы будем, если можем, начинать копировать из одного места.

Теперь просто для каждого ans[j] смотрим: если мы можем начать копировать оттуда же, откуда начали в ans[j], то оставляем ответ от ans[j], просто добавляя к первому числу в последней скобочке в нем на $i-j$, если нет, то просто приписываем новую скобку $(i-j,k)$, где k именно позиция той наибольшей строки, соответствующая суффиксу данного префикса.

Времени тратится $\mathcal{O}(n^2)$ - z-функция за n считается для каждого префикса, за n ищем ответ для каждого i.

5. ***Свертка***

Динамика по подотрезкам. Отрезки бывают двух типов: те, которые представляются в виде n(str) (если $n = 1$, то в виде str (заметим, что str это именно результат свертки, а не просто буквы)), либо там есть хоть одна свертка, которая упаковывает не всю эту строку.

Найти второе разложение можно перебором: переберем границу раздела, найдем свертку правой части, свертку левой, склеим их, получим свертку данной строки. Возьмем минимальну для всех разделов. Теперь как найти первое разложение. Посчитаем префикс-функцию этой строки. Как теперь проверить, что какой-то префикс строки постоянно повторяется, образуя всю строку? Если $p[i] = n - i$, то этот префикс, будучи домноженным на какое-то число, порождает всю нашу строку.
Рассмотрим нулевой символ. Он равен i-му, 1й будет равен i+1му, ... , i-1й будет равен 2i-1му. А теперь заметим, iй будет равен 2iму. В итоге получили, что такие символы будут равны:

0, i, 2i, 3i, ... , n-i
1, i+1, 2i+1, 3i+1, ... , n-i+1
...
i-1, 2i-1, 3i-1, ... , n-1

Т.е. строка представима в виде n(str), где $number = \frac{n}{i+1}$, а $str = s[0:i)$. Теперь заметим, что ответ для нашей строки - минимум из тех, что мы нашли разделением и ответа вида $\frac{n}{i+1}(go(s[0:i]))$. Если мы находимся в строке из одного символа, то ответ — этот символ. Если $\frac{n}{i+1} = 1$, то n приписывать не надо. Если минимум из всех тех ответов, что мы получили, длиннее нашей строки, то ответ — просто строка.

Оценка по времени: на каждом отрезке запоминаем ответ, так что $\mathcal{O}(n^3)$ (n^2 отрезков, на каждом ищем разделитель и префикс-функцию за n).

6. ***Быстрая покраска забора***

Посмотрим на самый левый элемент исходного массива. Понятно, что этот элемент мы могли покрасить самым первым и ответ бы никак не испортился. Нет смысла красить каким-то другим цветом этот элемент, если он потом будет закрашен, а слева от него все равно ничего нет. Поэтому можем считать, что он был покрашен в первую очередь.

Заметим: возможно, нулевой был покрашен не один, а ещё с кем-то. Переберем правую границу этого начально покрашенного отрезка (вполне может быть, что правая граница равна левой). Будем идти вправо и искать следующий элемент такого же цвета. Если не найдем, то от массива $0..n-1$ переходим к $1..n-1$ и увеличим ответ на 1. Если все же нашли, то скажем, что он был покрашен таким цветом в первую очередь, а все остальное уже потом на него накладывалось. То есть допустим, что в нулевой ячейке массива был записан цвет C_0 и мы нашли в клетке l тот же цвет C_0. Давайте считать, что отрезок $0..l$ был изначально покрашен в цвет C_0, а потом уже все остальное покрасилось.

Теперь $ans = min(ans, 1 + go(1, l-1, C_0) + go(l+1, n-1, C_0))$, где go вычисляет ответ на соответствующем подотрезке. Мы передаем ей C_0, чтобы на отрезке, от которого запустились, для других клеток с цветом C_0 тоже проверили, вдруг эти клетки тоже должны были тоже в первую очередь быть покрашены этим цветом. Так что на тех отрезках мы так же, как тут брали нулевой элемент и смотрели клетки с таким цветом, возьмем нулевую клетку этого отрезка и найдем клетки с равный ей цветом, прибавим 1 к ответу, запустимся от получившихся отрезков и так далее. Но на этом отрезке, в отличии от начального $0..n-1$ мы ещё найдем клетки цвета C_0, они будут делить отрезок на 2 части, запустимся от них (не прибавляя уже 1 к ответу). Так, пока не придем до отрезка длины 1, где ответ будет 0 или 1 в зависимости от того, какой нам цвет передали (равный или нет данному). Каждый раз, когда мы посчитали ответ для отрезка l, r, C_i, мы запоминаем его, чтобы 2 раза не считать ответ для одного отрезка.

Заметим теперь, что каждый отрезок окружен двумя цветами. Значит, в один отрезок l, r мы могли прийти только с двумя разными цветами (это будет просто зависит от того, в каком порядке мы их убирали).

Отрезков $\mathcal{O}(n^2)$. Каждый обрабатываем за $\mathcal{O}(n)$ (делаем по одному проходу для цвета, который передали и цвета, который в самой левой ячейке). Итоговая сложность — $\mathcal{O}(n^3)$. Корректность: я просто смотрю, какой отрезок раньше добавили. Почему можно смотреть от нулевого элемента написано в самом начале.