1 Задача A. Тимбилдинг

Решение на 25 баллов. Из-за ограничений $4 \le n \le 10$, $2 \le k < n$ и факта, что n делится на k, можно видеть, что возможно совсем немного различных вариантов входных данных, а именно, следующие пары (n, k): $(4, 2)$, $(6, 2)$, $(8, 2)$, $(10, 2)$, $(6, 3)$, $(9, 3)$, $(8, 4)$ и $(10, 5)$. На каждый из этих вариантов можно определить ответ, в буквальном смысле, посчитав его на листочке бумаги.

Решение на 50 баллов. Представим разбиение по командам в оба дня следующим графическим образом. Расположим всех n людей компании в виде таблицы из k строк и n/k столбцов таким образом, чтобы каждый столбец соответствовал одной команде, участвовавшей в первый день. Разбить людей на команды во второй день — значит покрасить клетки этой таблицы в n/k цветов так, чтобы каждого цвета было ровно k клеток (таким образом, каждый цвет соответствует одной команде второго дня).

Посчитаем количество пар людей, побывавших в одной команде в оба дня. В пределах одной команды первого дня, то есть, одного столбца, это количество выражается суммой $\frac{x_1(x_1-1)}{2} + \frac{x_2(x_2-1)}{2} + \ldots + \frac{x_{n/k}(x_{n/k}-1)}{2}$, где x_c — количество клеток цвета c в этом столбце. Заметим, что это выражение можно преобразовать как $\frac{1}{2}(x_1^2 + \ldots + x_{n/k}^2) + \frac{1}{2}(x_1 + \ldots + x_{n/k}) = \frac{1}{2}(x_1^2 + \ldots + x_{n/k}^2) + \frac{1}{2}k$, так как суммарное количество клеток в одном столбце — k.

Упражнение по математике: доказать, что минимум выражения $x_1^2 + \ldots + x_{n/k}^2$ при условии, что все x_i целые, и что $x_1 + \ldots + x_n = k$, достигается, если любые два числа x_i и x_j отличаются не более, чем на один (подсказка: иначе одно из этих чисел можно уменьшить на один, а другое — увеличить).

Таким образом, мы хотим, чтобы в каждом столбце каждого цвета было «приблизительно поровну». Несложно придумать, как нужно красить клетки таблицы, чтобы достичь такого эффекта — достаточно их красить сначала в первый цвет, потом во второй цвет, и так далее, в порядке следования по строкам таблицы. В данной группе таблицу можно явно сохранить построить в памяти программы, а потом честно посчитать количество пар людей, побывавших в одной и той же команде в оба дня.

11111
11222
22223
33333
34444
44455
55555

Рис. 1: Пример для $n = 35$, $k = 7$

Решение на 75 баллов. В этой группе нельзя посчитать искомое количество пар людей, явно перебрав все пары людей в построенном ответе, но можно это сделать, воспользовавшись, например, формулой из предыдущего пункта, по которой ответ считается за время, пропорциональное площади таблицы. Иными словами, в данной группе тестов требуется решение за время $O(n)$. Также, начиная с данной группы тестов ответ на задачу выходит за пределы 32-битного типа данных, поэтому для его хранения нужно воспользоваться 64-битным типом данных, предоставляемым вашим языком.

Решение на 100 баллов. В этой группе предлагается заметить, что ответ можно посчитать, явно не строя описанную таблицу в памяти. Действительно, если обозначить за $\%$ операцию взятия остатка от деления, а за div — операцию целочисленного деления, и положить $w = n/k$, $h = k$, то в каждом столбце будет ровно $h \% w$ цветов, в которые окрашены $(h \operatorname{div} w) + 1$ клеток, а все остальные цвета будут содержать по $h \operatorname{div} w$ клеток. Это позволяет записать явную формулу для ответа, для вычисления которой требуется константное количество арифметических операций.

2 Задача B. Большие планы

Решение на 50 баллов. В данной группе можно явно проверить требования миграционной политики для каждого отрезка поездки, состоящего из ровно b дней. Такое решение будет работать за сложность $O(a^2)$, чего достаточно для прохождения данной группы, но недостаточно для прохождения следующей.

Решение на 100 баллов. В данной группе тестов требуется так или иначе оптимизировать процедуру получения количества дней на данном отрезке времени, в течение которых турист находится на территории страны. Для этого можно, наример, составить массив префиксных сумм (т. е., массив $sum[i]$, в котором i-е значение равно количеству дней среди первых i дней, проведенных туристом в стране), и использовать выражения вида $sum[i+b] - sum[i]$ для подсчета количества проведенных тури-

стом дней в стране на произвольном отрезке времени. Таким образом, получается решение с временной сложностью $O(a)$.

3 Задача C. Чистые носки

Решение на 30 баллов. В данной группе тестов разрешается группировать только носки одинакового оттенка, поэтому достаточно поделить размер каждой группы носков одного оттенка пополам нацело, и просуммировать получившиеся значения.

Решение на 60 баллов. Данная подзадача предназначена для решений, которые могли бы при должной реализации получать сто баллов, но почему-то получают только шестьдесят :)

Решение на 100 баллов. Для решения данной задачи следует заметить, что в оптимальном ответе можно группировать только пары носков, чьи оттенки являются соседними в порядке сортировки всех оттенков носков. Отсортируем все носки в порядке возрастания их оттенка, и разобьем носки на группы, в каждой из которых расстояние между оттенками любых двух соседних носков не превосходит d. Нетрудно видеть, что сгруппировав соседние носки по парам в пределах каждой группы (возможно, один носок из группы останется невостребованным), мы получим оптимальный ответ. Решение имеет асимптотику $O(n \log n)$.

4 Задача D. Клавиатура и вирус

Чтобы решить эту задачу, нужно тщательно прочитать её условие, и понять, что в качестве ответа требуется посчитать максимально возможное количество элементов в пересечении множеств символов по всем парам языков.

Решение для 1 группы тестов. В данной группе можно перебирать символ первого языка и символ второго языка, и увеличивать счётчик ответа на один каждый раз, когда они совпадают.

Решение для 2 группы тестов. В данной группе можно завести массив размера 10^5, в котором увеличить на один все позиции, соответствуюшие элементам первого языка и элементам второго языка. Тогда те позиции, в которых в итоге окажется записана двойка, соответствуют элементам, лежащим в пересечении.

Решение для 3 группы тестов. В данной группе можно «сжать координаты» и воспользоваться первым решением, то есть, заменить каждое число из обоих множеств на его порядковый номер среди всех чисел. А можно отсортировать обе последовательности и посчитать количество элементов за линейное время, установив по указателю на первые элементы обоих последовательностей, и двигая тот из них на каждом шаге, который указывает на меньший элемент. Если в некоторый момент оба указателя соответствуют равным числам, то ответ можно увеличить на один.

Решение для 4 группы тестов. В данной группе можно перебрать все пары языков, и для каждой пары двумя вложенными циклами найти количество элементов в пересечении. Такое решение будет работать за время $O(n^2m^2)$.

Решение для 5 группы тестов. Представим каждое множество в виде 200-битной маски. Тогда количество элементов в пересечении двух языков — это количество установленных бит в побитовом «И» масок, соответствующих двум этим языкам. Будем хранить каждую 200-битную маску в виде 13 16-битных целых чисел. Тогда посчитать побитовое «И» двух масок можно за 13 битовых операций над соответствующими числами, а определить суммарное число бит в них можно, предподсчитав число установленных бит для всех возможных 16-битных целых чисел и сохранив результаты в массив. Таким образом, получается решение, совершающее порядка $10000^2/2 \cdot 13 \cdot 2$ очень простых биотвых операций, которое укладывается в ограничение по времени с трехкратным запасом.

Решение для 6 группы тестов. Разобьём символы на две категории: лёгкие и тяжёлые. Будем называть символ лёгким, если он встречается в менее, чем C языках, где C — некоторая константа, значение которой мы подберём позднее. Остальные символы будем называть, соответственно, тяжёлыми.

Идея решения такова. Заведём двумерный массив $ans[][]$, в котором будем считать размеры пересечений для каждой пары множеств. Для каждого лёгкого символа пройдёмся по всем парам множеств, в которых он фигурирует, и увеличим для них ответ на единицу. Все тяжелые символы перенумеруем

последовательными натуральными числами, и выпишем для каждого множества маску, в которой единицы стоят на позициях, соответствующих тяжёлым символам, присутствующим в множестве. После чего запустим на тяжёлых множествах решение из предыдущей группы тестов, использующее битовое сжатие.

Оценим количество тяжелых символов. Каждый тяжёлый символ встречается минимум C раз, а суммарный размер всех множеств — nm, значит тяжёлых символов — $\frac{nm}{C}$. Значит, сложность второй части решения — порядка $\frac{1}{32}\frac{n^2}{2}\frac{nm}{C} = \frac{1}{64}\frac{n^3m}{C}$.

Оценим время работы первой части решения. Первая часть решения работает за не более, чем $k_1^2 + \ldots + k_l^2$, где $k_1, \ldots, k_l$ — количество раз, которое встречается каждый из лёгких символов. При этом, у нас есть условия $0 \le k_i \le C$ и $\sum k_i \le nm$. Можно показать, что максимум данного выражения достигается, когда все k_i кроме, возможно, одного, равняются нулю или C (идея: если у нас есть два числа k_i и k_j строго между нулём и C, то их можно «раздвинуть» с сохранением суммы, при этом сумма их квадратов увеличится). Значит, количество операций в первой части решения имеет порядок $\frac{C^2}{2}\frac{nm}{C} = \frac{nmC}{2}$.

Значит, общая сложность решения — $\frac{nmC}{2} + \frac{1}{64}\frac{n^3m}{C} \ge 2\sqrt{\frac{nmC}{2} \cdot \frac{1}{64}\frac{n^3m}{C}} = \frac{n^2m}{4\sqrt{2}}$ (неравенство следует из неравенства между средним арифметическим и средним геометрическим), а минимум достигается при $C = \sqrt{\frac{n^2}{32}}$.

Заметим, что это лишь теоретическая оценка, позволяющая оценить асимптотику решения, а на практике же следует подобрать оптимальное значение константы C экспериментально. В зависимости от оптимальности реализации и аккуратности подбора константы C, такое решение набирает на данной группе тестов от 0 до 40 баллов.

5 Задача E. Dungeons & Dragons

Решение на 30 баллов. В данной подзадаче можно перебрать как значение, выпавшее на первом кубике, так и значение, выпавшее на втором кубике, и проверить, что их сумма в действительности равна n.

Решение на 60 баллов. В данной подзадаче предлагается заметить, что перебирать значение, выпавшее на втором кубике, не нужно, так как оно вычисляется по первому значению по формуле $y = n - x$, где x — значение, выпавшее на первом кубике, а значит, его можно посчитать, и проверить, что оно принадлежит диапазону от c до d.

Решение на 100 баллов. Чтобы решить данную подзадачу, проще всего воспользоваться геометрическим представлением для задачи. Пусть x — число, выпавшее на первом кубике, а y — число, выпавшее на втором кубике. Видно, что множество всевозможных положений точки (x, y) на плоскости описывается прямоугольником, простирающимся по горизонтали от a до b, и по вертикали от c до d. Нас, в действительности, спрашивают количество целых точек, лежащих в пересечении данного прямоугольника и прямой $y = x - n$. Воспользовавшись данным чертежом, можно увидеть, что принципиально возможно пять вариантов взаимного расположения прямой и прямоугольника (в зависимости от количества вершин прямоугольника, попавших ниже прямой), а также можно составить для каждого из этих вариантов простую формулу, описывающую ответ в соответствующем случае. Например, если $n < a + c$, то нетрудно видеть, что ответ — 0, а если $a + c \le n \le \min\{a + d, b + c\}$, то ответ равен $n - a - c + 1$. Оставшиеся случаи остаются пытливому читателю как упражнение.

6 Задача F. Дураки и дороги

Решение на 30 баллов. Будем говорить, что ребра окрашены в цвета, в зависимости от того, какая компания ими владеет. В данной подзадаче есть ребра всего двух цветов, поэтому задача на самом деле формулируется следующим образом: правда ли, что есть путь из первой столицы во вторую, содержащий рёбра только первого цвета? А второго?

Для ответа на оба этих вопроса можно составить граф только из рёбер первого и второго цвета соответственно, и воспользоваться любым алгоритмом поиска пути в нём (например, обходом в ширину

или в глубину).

Решение на 60 баллов. В данной подзадаче предлагается развить идею предыдущего решения, а именно, для того, чтобы понять, что цвет c является обделенным, надо составить граф из рёбер всех цветов кроме c, и проверить, связны ли в нём вершины 1 и 2.

Решение на 100 баллов. Данная подгруппа решается с помощью подхода, именуемого Divide-and-Conquer (что переводится как Разделяй-и-Властвуй, а в оригинале звучит как Divide Et Impera). Сформулируем следующую рекурсивную процедуру:

```
G = текущий граф, хранимый в некотором виде
% к моменту входа в данную процедуру в графе есть ребра
% всех цветов, кроме отрезка цветов [l, r].
process(l, r):
| if (l == r):
| | % в графе сейчас есть все ребра кроме цвета l,
| | % значит можно ответить на вопрос для него!
| | answer[l] = связны ли вершины 1 и 2 сейчас в графе G
| else:
| | m = (l + r) / 2
| | добавь все рёбра цветов l..m в граф G
| | process(m + 1, r)
| | откати все изменения в графе G, произведённые двумя строками выше
| | добавь все рёбра цветов (m+1)..r в граф G
| | process(l, m)
| | откати все изменения в графе G, произведённые двумя строками выше
```

Для решения задачи, надо вызвать данную процедуру с параметрами 1 и k. Можно убедиться в корректности данной процедуры, проверив, что в каждом рекурсивном вызове выполнено пред-условие, описанное в комментарии перед ней. Осталось понять, в каком именно виде следует поддерживать граф, чтобы описанная процедура работала эффективно. Оказывается, если держать вершины графа в DSU (disjoint set union, система непересекающихся множеств), и аккуратно откатывать все изменения, которые происходят с данной структурой (например, записывая все производимые изменения в стек), то данная процедура работает за время $O(n \log k \log n)$! Действительно, можно показать, что каждое ребро будет добавлено в граф в не более, чем $\log k$ рекурсивных вызовах, а время работы системы непересекающихся множеств с эвристикой сжатия путей составляет $O(\log n)$ на запрос.

Данное решение является упрощенным частным случаем техники Dynamic Connectivity Offline. Данную задачу можно также решать, сводя ее к стандартному варианту Dynamic Connectivity Offline, но более общий подход может оказаться менее эффективным в силу повышенного потребления памяти (от $O(m)$ до $O(m \log m)$ в зависимости от реализации). Подробнее о более общей задаче можно прочитать в данной статье: `http://se.math.spbu.ru/SE/diploma/2012/s/Kopeliovich_diploma.pdf`.

7 Задача G. Очередь в столовой

Идея любого разумного решения. В первую очередь следует понять, как устроен процесс расширения очереди. Заметим, что если в очереди было x человек, и между каждыми двумя соседями вклинилось по a человек, то итоговый размер очереди станет равен $x' = x + (x-1)a$. Это можно переписать в виде равенства $x' - 1 = x - 1 + (x-1)a = (x-1)(a+1)$. Таким образом, если была произведена цепочка вклиниваний из $a_1, a_2, \ldots, a_k$ человек соответственно, то верно равенство $x' - 1 = (2-1)(a_1+1)(a_2+1)\ldots(a_k+1) = (a_1+1)(a_2+1)\ldots(a_k+1)$, а значит, задача состоит в том, чтобы представить число $n-1$ в виде максимального количества сомножителей. Как известно из элементарной теории чисел, для этого нужно представить число $n-1$ в виде произведения простых чисел, то есть, факторизовать его.

Задачу факторизации можно делать множеством разных способов. Элементарный и достаточно эффективный метод факторизации за $O(\sqrt{n})$ набирает уже 80 баллов. Для получения ста баллов можно воспользоваться либо методом Лемана, либо методом Полларда факторизации целого числа.

8 Задача H. Путник

Решение на 30 баллов. В данной подгруппе можно перебрать все варианты того, какие остановки делает электричка, и выбрать из них оптимальный.

Решение на 60 баллов. Будем двигаться от начала пути, вычисляя следующую величину методом динамического программирования. Пусть $D[i][t]$ — это максимальное количество пассажиров со станций с номерами не больше i, которые поедут на нашей супер-электричке, если от станции i до конца маршрута электричка проедет за время t. Нетрудно выразить $D[i][t]$ через $D[i-1][t-p_i]$ и $D[i-1][t-p_i-1]$, в зависимости от того, останавливается ли электричка на остановке i или нет.

Решение на 100 баллов. Будем двигаться от начала пути, вычисляя следующую величину методом динамического программирования. Пусть $D[i][j]$ — это максимальное количество пассажиров со станций с номерами не больше i, которые поедут на нашей супер-электричке, если на этих станциях электричка суммарно сделает j остановок. Чтобы посчитать $D[i][j]$ через $D[i-1][j-1]$ и $D[i-1][j]$ заметим, что возможно два варианта: электричка может как пропустить остановку i (в таком случае значение будет просто равно $D[i-1][j]$), так и остановиться на ней (в таком случае значение будет равно $D[i-1][j-1]$ плюс, возможно, c_i, если наша супер-электричка с учётом j остановок доезжает до пункта назначения строго быстрее старой электрички). Получаем решение с временной сложностью $O(n^2)$.

9 Задача I. Рефераты

Решение на 15 баллов. В данной подгруппе можно было реализовать ровно то, что написано в условии: перебрать для каждой строки все её подстроки, и найти из них самую короткую, которая не входит ни в одну другую как подстрока. Всего мы рассмотрим $O(L^2)$ подстрок, каждую из которых нужно приложить к остальным строкам в $O(L)$ местах, и проверить на совпадение за время $O(L)$. Таким образом, наивная реализация этого алгоритма работает за время $O(L^4)$ с очень маленькой константой и с лёгкостью укладывается в ограничение по памяти.

Решение на 30 баллов. В данной подгруппе можно захешировать все подстроки каждой строки, и для каждого хеша в хеш-таблице сохранить, в какой строке он встречается, либо пометить его как плохой, если он встречается более, чем в одной строке. После этого можно перебором для каждой строки найти её кратчайшую подстроку, которая не является плохой строкой, либо определить, что такой нет. Сложность такого решения — $O(L^2)$.

Решение на 45 баллов. Заметим, что для каждой строки можно найти самую длинную её подстроку, не встречающуюся в других строках, с помощью бинарного поиска (действительно, если в строке есть такая подстрока длины l, то и для любой длины больше l такая подстрока найдётся, достаточно расширить подстроку длины l до нужного числа символов). Значит, для каждой строки сделаем бинарный поиск по длине искомой подстроки, после чего выпишем хеши всех подстрок всех строк зафиксированной длины, и проверим, есть ли среди них такой хеш, который встречается только в нашей строке. Такое решение работает за время $O(nL\log L)$.

Решение на 60 баллов. Данную подгруппу можно пройти вариацией предыдущего решения, использующей идею корневой оптимизации. Заметим, что в качестве подзадачи в предыдущем решении возникает процедура составления хеш-таблицы из хешей всех подстрок длины l всех строк. Предподсчитаем все такие хеш-таблицы для всех длин не более A, где A — некоторый параметр, значение которого мы выберем позднее. Мы можем сделать это за время $O(LA)$, и обращаться к ним впоследствии по необходимости. Если же в процессе требуется обратиться к хеш-таблице для большей длины, то полностью построим её, как и в предыдущем решении. Однако заметим, что к такой процедуре нам придётся прибегнуть только для «тяжёлых строк», то есть, для таких строк, у которых длина $|s_i|$ превосходит A. А таких строк всего может быть не более $\frac{L}{A}$ (так как иначе их суммарная длина превзойдёт L). Значит, общее время работы решения составляет $O(LA)$ на предподсчёт и $O(\frac{L}{A}L\log L)$ на обработку всех строк. Положив $A \sim \sqrt{L\log L}$, получим решение со временем работы $O(L\sqrt{L\log L})$, что проходит ограничения данной группы.

Решение на 80–100 баллов. В последних двух группах требуется прибегнуть к одной из распространённых структур данных, предназначенных для работы с подстроками: суффиксному массиву,

суффиксному дереву или суффиксному автомату. Мы опишем решение с помощью суффиксного массива, как наиболее простое для восприятия.

Построем суффиксный массив для конкатенации всех исходных строк через некоторый разделитель. Построим по этому суффиксному массиву с помощью алгортима Касаи-Аримуры-Арикавы-Ли-Парка массив наибольших общих префиксов $LCP[]$, а также построим над этим массивом разреженные таблицы (sparse-table) для ответа на запрос минимума на отрезке за время $O(1)$. Сопоставим каждому элементу суффиксного массива номер исходной строки, чьим суффиксом он является.

Рассмотрим некоторую позицию в суффиксном массиве. Эта позиция соответствует суффиксу какой-то их исходных строк. Определим для этого суффикса, какой самый короткий его префикс удовлетворяет условию задачи, то есть, встречается только в строке, соответствующей этому суффиксу. Для этого надо найти максимальный возможный наибольший общий префикс данного суффикса со всеми суффиксами, соответствующими оставшимся строкам набора. В силу свойства, что наибольший общий префикс двух суффиксов в суффиксном массиве равняется минимуму на отрезке массива $LCP[]$, понятно, что этот максимум достигается на ближайшей к текущему суффиксу позиции, соответствующей другой строке набора, либо до, либо после текущего суффикса. Посчитаем номера этих двух ближайших позиций, и воспользуемся разреженными таблицами для вычисления минимума на отрезке.

Таким образом, для каждого суффикса каждой строки мы знаем, какой минимальной длины у него надо взять префикс, чтобы получившаяся подстрока удовлетворяла условию задачи, а значит, мы знаем и ответ на задачу.

Подобное решение работает за время $O(L \log L)$, и, в зависимости от аккуратности реализации, набирает либо 80, либо 100 баллов. Возможны также и линейные решения с помощью других суффиксных структур.

10 Задача J. Новогодний и прямоугольный

Решение на 10 баллов. Можно перебрать все возможные прямоугольники и выбрать, например, самый маленький из тех, в которых находится больше всего мандаринок.

Решение на 30 баллов. Переберём все прямоугольники с нижним левым углом в нижнем левом углу поля. Тогда, очевидно, самый нижний-левый из тех, в которых есть хотя бы один апельсин, задаёт позицию левого нижнего угла искомого прямоугольника, а самый нижний-левый из тех, в которых больше всего апельсинов, задаёт позицию верхнего правого угла искомого прямоугольника.

Решение на 50 баллов. Спросим про все прямоугольники вида $(1, 1)$–(t, n). До некоторого момента ответы на запросы будут равны нулю, начиная с некоторого момента они станут линейно расти, и с некоторого момента они станут снова постоянны и равны S, где S — размер искомого прямоугольника. Два описанных момента задают левую и правую границы искомого прямоугольника. Аналогично найдем верхнюю и нижнюю границы искомого прямоугольника.

Решение на 75 баллов. Будем искать каждый из четырех описанных в предыдущем решении моментов с помощью бинарного поиска. Количество запросов, сделанных таким решением, не превосходит $4 \log n + 4 \leq 128$.

Решение на 100 баллов. Узнаем первым запросом искомую площадь прямоугольника S. Найдем верхний правый угол прямоугольника (x, y) с помощью двух бинарных поисков, как в предыдущем решении. Сделав запрос в $(x-1, y)$, мы получим некоторую величину S'. Заметим, что искомая высота прямоугольника h может быть вычислена как $S - S'$, а значит, и искомая ширина w может быть вычислена после этого как S/h. Если аккуратно оценить количество запросов, производимое данным решением, то окажется, что оно не превосходит $2 \log n + 2 \leq 64$.

11 Задача K. Михаил наносит ответный удар

Решение на 20 баллов. Данная группа решается различными переборами возможного ответа (например, можно перебрать все строки длины не больше 8 из букв ‘a’ и ‘b’).

Последующие группы решаются с помощью различных вариаций некоторой задачи динамического программирования, имеющей квадратичную асимптотику. Например, можно заметить, что минималь-

ное количество букв, которое надо добавить в строку, чтобы она стала палиндромом, равно разности длины строки и длины максимальной её подпоследовательности, являющейся палиндромом. Значит, задача сводится к подсчету длины максимальной подпоследовательности, которая является палиндромом. Однако, в отличие от классических задач, в данной задаче установлено очень маленькое ограничение по памяти, в результате чего количество баллов сильно зависит от потребляемой памяти, которая существенно расходуется в момент восстановления ответа.

Решение на 35–50 баллов. Рассмотрим следующую величину: $D[l][r]$ — это длина максимальной подпоследовательности подстроки с l-го символа по r-й, которая является палиндромом. Нетрудно видеть, что $D[x][x] = 1$, $D[x][x-1] = 0$ (начальные условия), а также

$$D[l][r] = \begin{cases} \max\{D[l+1][r], D[l][r-1]\}, & S[l] \neq S[r] \\ \max\{D[l+1][r], D[l][r-1], 1 + D[l+1][r-1]\}, & S[l] = S[r] \end{cases}$$

В зависимости от аккуратности реализации, такое решение может набрать как 35 баллов, так и 50, так как для восстановления ответа в простейшем варианте требуется сохранение всей матрицы $D[][]$.

Решение на 70–100 баллов. Сохраним не все строки матрицы $D[i][j]$, а только те, номера которых делятся на k. Тогда для того, чтобы обратиться к произвольной строке этой матрицы, нужно «досчитать» её от последнего запомненного момента, на что уходит в худшем случае $O(kn)$ времени и $O(kn)$ дополнительной памяти. Таким образом, ценой увеличения времени работы в два раза, мы сократили расход памяти до $O(kn + \frac{n}{k}n)$, что, путём выбора константы k пропорциональной $\sqrt{n}$, приводится к виду $O(\sqrt{n})$.

Данное решение можно рассматривать как двухуровневую схему оптимизации памяти в два раза по порядку, ценой увеличения времени работы в два раза. Рассматривая аналогичные трёхуровневые, четырёхуровневые, и больших порядков схемы, можно довести это решение до $O(n \log n)$ потребляемой памяти ценой увеличения времени работы в $O(\log n)$ раз.

Решение на 100 баллов. Посчитаем величину $L[i][j]$, равную наибольшей общей подпоследовательности префикса длины i строки S и префикса длины j развернутой строки S'. Предположим, что искомый палиндром имеет нечётную длину. Тогда заметим, что центр искомого палиндрома c обладает тем свойством, что $L[c][n-c+1] = 1 + ans$, где ans — это длина искомого ответа. Значит, мы умеем сводить нашу задачу к поиску наибольшей общей подпоследовательности двух строк с использованием малого количества памяти. Данная задача решается с помощью Divide-and-Conquer с линейным количеством памяти с использованием алгоритма Хиршберга.

12 Задача L. Космические захватчики

Данная задача оценивалась потестово, поэтому в ней доступен полный простор для написания решений разной степени оптимальности. Самым прямолинейным способом решить эту задачу является построение всех точек пересечения и выделение граней в планарном графе, с их последующим подсчётом, что при аккуратной реализации за квадратичное время набирает 60 баллов.

В случае, когда никакие две окружности не касаются, можно предложить следующий алгоритм: будем добавлять окружности по одной. Пусть новодобавленная окружность пересеклась с k уже нарисованными. Тогда можно показать, что после её добавления количество областей, на которые делится плоскость окружностями, увеличивается на $k+1$. Таким образом, нужно при добавлении каждой окружности сказать, со сколькими уже нарисованными она пересекается. Это можно сделать за $O((n+m)^2)$ наивно, либо за $O((n+m)\log(n+m))$ или $O(n+m)$, оформив условие, что окружности пересекаются, в виде двойного неравенства, включающего в себя их радиусы и расстояние между их центрами, и воспользовавшись бинарным поиском или методом двух указателей. Такое решение набирает от 30 до 70 баллов, в зависимости от эффективности реализации. К сожалению, такое решение тяжело применить в случае, когда некоторые пары окружностей касаются, поэтому мы предложим более общий способ, как можно преодолевать подобные трудности.

Представим изображение на плоскости в виде планарного графа, в котором вершины бывают трёх видов:

1. Точки касания окружностей.

2. Точки пересечения каждой окружности с линией центров окружностей, не являющиеся точками первого типа.

3. Точки пересечения окружностей, не являющиеся точками первого типа.

Рёбрами данного графа будем считать дуги окружностей, соединяюшие вершины. Обозначим количество вершин за V, количество рёбер за E, количество граней за F и количество компонент связности данного графа за C. Тогда верна формула Эйлера, связывающая эти четыре величины: $V - E + F = 1 + C$.

Обозначим за a количество вершин первого типа, за b количество вершин второго типа и за c количество вершин третьего типа. Заметим, что каждая вершина первого типа имеет степень, равную четырём, каждая вершина второго типа имеет степень, равную двум, и каждая вершина третьего типа имеет степень, равную четырём. Значит, можно записать два уравнения:

$$\begin{cases} V = a + b + c \\ 2E = 4a + 2b + 4c \end{cases}$$

(второе равенство следует из факта, что удвоенное количество рёбер в графе есть сумма степеней всех вершин).

Оставим в качестве упражнения читателю нахождение величин a, b и c за линейное время с помощью метода двух указателей или чуть менее эффективно с помощью бинарного поиска. Таким образом, нам достаточно научиться считать C, чтобы из формулы Эйлера мы могли выразить требуемую величину F. Это также можно сделать за линейное время с помощью следующего (жадного) алгоритма. Заметим, что любая компонента связности представляет собой объединение некоторого количества окружностей первого типа с некоторым количеством окружностей второго типа. Более того, множество окружностей первого типа в компоненте связности представляет собой непрерывный отрезок окружностей в порядке сортировки по радиусу, и аналогичное верно для окружностей второго типа.

Это позволяет сформулировать следующий жадный алгоритм: рассмотрим среди оставшихся к данному моменту окружностей ту, которая имеет наименьший радиус. Найдём все окружности, с которыми она пересекается, из них выберем ту, которая имеет максимальный радиус, и пометим все меньшие, как принадлежащие очередной компоненте связности. Из окружностей, пересекающих эту максимальную, снова выберем ту, которая имеет максимальный радиус, и пометим все меньшие её, и так далее. Таким образом, мы можем выделить очередную компоненту связности за время, пропорциональное её размеру. Значит, мы можем за линейное время посчитать количество компонент связности, а значит, и требуемое количество граней.

Несмотря на кажущуюся громоздкость объяснения данного решения, рекомендуется усвоить идею с использованием формулы Эйлера, так как она обобщается на случай подсчёта областей при разбиении плоскости кривыми произвольной формы.