Задача 1. Ускорение

Пусть $T' = \frac{T}{x}$, тогда если $t \leqslant T - T'$ — ответ `-1`, так как, чтобы получить ускорение в x раз, время на которое необходимо сократить программу, больше, чем время исполнения известной Диме части кода.

Иначе составим уравнение: $\frac{T}{x} = T - t + \frac{t}{ans}$, из которого выражаем ответ: $ans = \frac{t}{(\frac{T}{x} - T + t)}$.

Задача 2. Китайская грамота

Для решения данной задачи необходимо при считывании всей строки по одному символу хранить каждый i-ый символ в отдельном контейнере, например для языка python подошел бы список, а языка C++ — `vector <char>`. После считывания строки и заполнения списков символов необходимо было вывести все получившиеся строки, даже если они были пустыми.

Задача 3. Жюри готовят задачи

Поскольку все члены жюри ждут, пока не соберутся все, самое раннее время начала — это время, когда освободится последний член жюри. Для того чтобы найти это время, нужно для каждого посчитать, когда он закончит выполнять свои дела.

Рассмотрим два случая: когда член жюри завершает работу до того, как приедет первый автобус и после этого.

В первом случае, член жюри сразу садится на автобус, а значит время, когда он закончит, будет равно $t_2 + t_4 + t_5$.

Во втором случае жюри придётся подождать какое-то время. Переведем все время в минуты от начала первого дня. Тогда время отъезда будет равно $\left\lceil \frac{t_1 - t_2}{t_3} \right\rceil \times t_3$, а окончательное время будет равно $\left\lceil \frac{t_1 - t_2}{t_3} \right\rceil \times t_3 + t_4 + t_5$.

При желании, эту формулу можно объединить в одну: $\max\left(\left\lceil \frac{t_1 - t_2}{t_3} \right\rceil, 0\right) \times t_3 + t_4 + t_5$.

Задача 4. Очень скучная головоломка, абсолютно сбивающая темп контеста

Первое, что должно приходить в голову при решении любой задачи, это: «А не попробовать ли мне решить её перебором?». Ну что ж, давайте попробуем перебрать решения, однозначно задав состояние каждой трубе, а затем проверить, есть ли путь от стартовой до конечной трубы.

Тогда количество всех перебираемых состояний поля будет равно: $2^I \cdot 4^L$, где, где I — число прямых труб, L — число угловых труб. Для проверки существования пути нам придётся в худшем случае пройти все клетки поля, а также клетки старта и конца (округлим до $n \cdot m$).

Тогда асимптотическая сложность решения будет равна произведению числа состояний, на количество клеток, которые необходимо пройти для проверки существования пути, то есть $O(n \cdot m \cdot (2^I \cdot 4^L))$. Очевидно, что такое решение не пройдёт ни одну подгруппу, поэтому думаем дальше.

Можно заметить, что головоломка с трубами — это просто лабиринт с недетерминированными (неопределёнными заранее) состояниями клеток. А любой лабиринт вполне можно представить в виде графа, в котором непременно можно найти путь. Выше мы уже поняли, что мы не можем перебирать все состояния, так как их слишком много. Зато мы можем перебирать все пути, так как клетка с прямой трубой никак не увеличивает число возможных путей в лабиринте. Это происходит благодаря тому, что положение прямой трубы при прохождении клетки определяется однозначно (по направлению взгляда), в отличие от угловой трубы, которая имеет два возможных положения относительно направления взгляда (повёрнута влево или вправо от направления взгляда), что увеличивает число путей при прохождении клетки с трубой примерно в два раза (данное число является очень грубым приближением и в реальности число порождаемых путей будет меньше, но для простоты вычислений мы будем считать именно так). Тогда всего надо будет перебрать не более $2^L = 1024$ путей. Для этого воспользуемся алгоритмом поиска в глубину (Breadth First Search или

же BFS), каждый из которых пройдёт не более $n \cdot m$ клеток поля. Соответственно асимптотика будет $O(2^{10} \cdot n \cdot m)$ или же в худшем случае примерно $O(10^7)$, что позволяет нам решить эту задачу.

Остался только один вопрос, как же перебирать эти пути?

Начнём проход от старта и будем передавать в очередь нашего BFS следующую информацию: направление взгляда, текущие координаты, пройденное расстояние, множество координат уже посещённых вершин. Если мы находимся на клетке с прямой трубой, то идём по направлению взгляда в следующую клетку (при условии, что мы в ней ещё не были). Если труба угловая, то перебираем пути, идущие направо и налево от взгляда (при тех же условиях, что и у прямой трубы).

Для решения только части подгрупп можно было разработать более простые решения, составление которых оставим вам, как упражнение.

Задача 5. Пишущая головка

Подзадача 2

Эту подзадачу можно было решить, сделав ровно то, что описано в условии. Создать матрицу. Заполнить ее значениями, используя базовый паттерн. Отвечать на запросы, поэлементно суммируя значения на нужных прямоугольниках. Асимптотика решения — $O(N^2 \cdot Q)$.

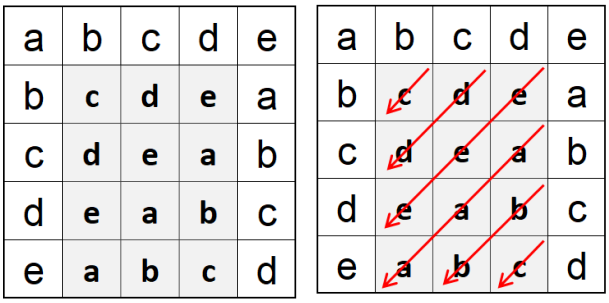
Подзадача 3

Для этой подзадачи нужно было использовать префиксные суммы: либо двумерные (с асимптотикой решения $O(N^2 + Q)$), либо одномерные префиксные суммы в каждой строчке, что тоже заходило (асимптотика $O(N^2 + N \cdot Q)$).

Подзадача 4

В этой подзадаче ограничения были выставлены таким образом, чтобы сформировать полную матрицу мегапикселя было невозможно (ограничение 256 мегабайт на память). Поэтому нужно было придумать что-то другое.

Для объяснения идеи давайте сформируем тестовую матрицу с базовым паттерном a, b, c, d, e (числа заменены на буквы для наглядности). И, для примера, посмотрим на нее с точки зрения запроса 2 2 5 4.



На второй картинке красными стрелочками обозначено интересное наблюдение. Дело в том, что каждая стрелочка проходит над одинаковыми числами (буквами). Также количество таких стрелочек не превосходит $2 \cdot N - 1$, так как стрелочка начинается либо в первой строке, либо в последнем столбце прямоугольника. Также можно заметить, что длины стрелочек изменяются довольно понятным образом. До какого-то момента они постепенно увеличиваются, причем каждый раз на единицу, затем сохраняют свою длину, затем последовательно уменьшаются, также на единицу за один шаг. Причем данная последовательность как начинается со стрелочки длины один, так и заканчивается на стрелочку длины один.

Давайте попробуем посчитать сумму на каждой стрелочке, а потом сложить эти суммы. Тогда, если мы будем считать сумму на стрелочке за $O(1)$, то на запрос будем отвечать за $O(2 * N - 1) = O(N)$.

Это сделать очень легко. Сумма на стрелочке — это просто её длина, умноженная на число, записанное под ней. Длины стрелочек легко получаются из условий, что последовательность их длин начинается и заканчивается на единицу. Также можно легко вывести, что максимальная

возможная длина стрелочки — $min(width, height)$, где $width$ и $height$ ширина и высота прямоугольника соответственно. Вот и все! Асимптотика этого решения все еще $O(N \cdot Q)$, но зато теперь нам не нужно хранить огромную матрицу в памяти.

Подзадача 5

В этой подзадаче нужно было развить идею предыдущей. Подытожим, что мы должны найти сумму вида $a+2b+3c+4d+4e+4f+3g+2h+i$, здесь обозначения букв уже произвольные (множители также будут зависеть от конкретного прямоугольника, такие значения взяты просто для наглядности). Эту сумму можно разделить на три другие. «Возрастающий» кусочек — $a+2b+3c$. «Константный» кусочек — $4d + 4e + 4f$. «Убывающий» кусочек — $3g + 2h + i$. Сразу понятно, что «константный» кусочек можно посчитать с помощью префиксных сумм. Что же делать с двумя другими? На самом деле, их тоже можно подсчитать, используя модифицированные префиксные суммы. Рассмотрим это на примере «возрастающего» кусочка.

Посчитаем обычные префиксные суммы, причем в обратном порядке. Посчитаем префиксные суммы префиксных сумм, также в обратном порядке.

Длина массива для упрощения равна 5			
index	arr	prefs	prefpref
0	a	a + b + c + d + e	a + 2b + 3c + 4d + 5e
1	b	b + c + d + e	b + 2c + 3d + 4e
2	c	c + d + e	c + 2d + 3e
3	d	d + e	d + 2e
4	e	e	e
5		0	0

Из этих массивов теперь можно выразить нужную нам сумму по следующей формуле: $incsumm(l, r) = prefpref[l] - prefpref[r + 1] - (r - l + 1) \cdot pref[r + 1]$.

К примеру, $incsumm(1, 3) = b + 2c + 3d = prefpref[1] - prefpref[4] - 3 \cdot pref[4] =$
$= b + 2c + 3d + 4e - e - 3e = b + 2c + 3d$.

Или, например, $incsumm(0, 1) = a + 2b = prefpref[0] - prefpref[2] - 2 \cdot pref[2] =$
$= a + 2b + 3c + 4d + 5e - (c + 2d + 3e) - 2(c + d + e) = a + 2b$.

Аналогично можно посчитать сумму на «убывающем» кусочке (в этом случае префиксные суммы нужно считать в прямом порядке). Таким образом, мы найдем необходимую сумму всех кусочков за $O(1)$. И решение всей задачи будет работать за $O(N + Q)$.

Задача 6. В бухгалтерии опять всё перепутали

Подзадача 2

Для решения второй подзадачи достаточно было написать $dp[begin][end]$, где $begin$ и end — начало и конец подстроки, а значение динамики — наибольший номер этой строки в окончательном списке. Соответственно, хотим максимизировать значения. Для пересчета можно было перебрать начало предыдущей строки $oldbegin$, и, если $s[oldbegin..(begin-1)]$ лексикографически меньше, чем $s[begin..end]$, обновить $dp[begin][end] = max(dp[begin][end], dp[oldbegin][begin - 1] + 1)$. Сравнение строк можно производить за линию. Для восстановления ответа можно использовать массив родителей. Таким образом, решение работает за $O(N^4)$.

Подзадача 3

Для решения третьей подзадачи достаточно было сделать все то же самое, что для второй, за исключением линейного сравнения строк. Для этого до подсчета динамики можно было предподсчитать массив $lcp[start1][start2]$ — длина наибольшего общего префикса подстрок s, которые начинаются

в символе $start1$ и $start2$, соответственно. Находить наибольший общий префикс можно также за линию. Эта часть отработает за $O(N^3)$.

Далее считаем динамику тем же самым образом, только теперь мы можем сравнивать строки за $O(1)$, т.к. мы знаем их наибольший общий префикс (можно посмотреть на символы, следующие сразу за наибольшим общим префиксом, и понять, какая строка больше). Таким образом, динамика работает также за $O(N^3)$. И общая асимптотика $O(N^3)$.

Подзадача 4

Во-первых, так как в ограничения по времени $O(N^3)$ не влезет, нужно переписать предподсчет lcp, используя простое динамическое программирование: $lcp[i][j] = (s[i] == s[j]\ ?\ lcp[i+1][j+1]+1 : 0)$.

Далее, пусть $dp[start][len]$ — наибольший номер строки $s[start..(start+len-1)]$ в окончательном списке. В этом случае пересчет динамики изменится следующим образом. При обновлении динамики в первую очередь будем перебирать $start$ — начало новой строки. Затем будем перебирать $oldstart$ — начало предыдущей строки. Обозначим $oldlen = start - oldstart$ — длину предыдущей строки, а $lcp = lcp[start][oldstart]$ — наибольший общий префикс предыдущей и следующей строки. Тогда имеем следующий пересчет:

- Если $oldlen \leqslant lcp$, то в качестве длины следующей строки мы можем взять любое число строго большее чем $oldlen$. Потому что если возьмем длину равную $oldlen$, следующая и предыдущая строки будут равны. А если возьмем длину меньшую $oldlen$, то и вовсе, следующая строка будет меньше предыдущей.
- Иначе же, если $oldlen > lcp$ и строка, которая начинается в $start$ лексикографически больше строки, которая начинается в $oldstart$, то в качестве длины следующей строки мы можем взять любое число, строго большее чем lcp. Потому что, если возьмем число меньше либо равное lcp, то следующая строка будет лексикографически меньше предыдущей, так как длина предыдущей $> lcp$, а длина следующей $\leqslant lcp$.
- В остальных случаях, когда $oldlen > lcp$ и строка, которая начинается в $start$ лексикографически меньше строки, которая начинается в $oldstart$, никакую длину новой строки мы взять не можем. Доказывается аналогично двум предыдущим утверждениям.

Заметим, что при обновлении динамики мы использовали только обновление типа «для всех значений строго больших X», причем во время расчета $dp[start][...]$ мы обращаемся только к элементам $dp[oldstart][...]$, а $oldstart < start$ (так как мы не рассматриваем пустые строки). Из этого можно сделать вывод, что «финализировать» значения $dp[start][...]$ можно только в конце итерации цикла по $start$, т.к. внутри пересчета $dp[start][...]$ они ни на что не влияют. А, так как в качестве обновления мы используем операцию $max=$ на суффиксе, то на этапе пересчета $dp[start][...]$ можно просто заполнять $dp[start][X+1]$ для всех X (только в точке, не на всем суффиксе!), а в конце каждой итерации цикла по $start$ обновить все значения $dp[start][...]$ пройдясь префиксным максимумом. Можно увидеть, что это даст ровно те же результаты, как если бы мы каждый раз выполняли $max=$ на всем суффиксе.

Асимптотика этого решения — $O(N \cdot (N+N)) = O(N^2)$, так как внутри цикла по $start$ мы вначале перебираем $oldstart$, и только после этого проходимся префиксным максимумом. Уже в этой подзадаче могли начаться проблемы с ограничением по памяти. Для их решения можно было использовать тип $short$ для хранения матриц или восстанавливать ответ без использования массива предков.

Подзадача 5

Эта подзадача заставляла сокращать объем памяти, потребляемый вашей программой. Было довольно много способов сделать это. Например использовать тип short и хранить только верхние/нижние треугольники матриц (можно заметить что вторые половины не используются). Или построить суффиксный массив с разреженными таблицами, чтобы находить lcp без матрицы $lcp[][]$.

Либо вообще не предподсчитывать lcp, а вычислять его в каждой итерации цикла по $start$ с помощью z-функции. Это возможно, т.к. в каждой в итерации цикла по $start$ мы смотрим только на $lcp[start][...]$.

Задача 7. Чапаев и Зебрино

Для первой подгруппы тестов достаточно найти расстояние между начальной и конечной точкой.

Для второй подгруппы можно использовать вложенный тернарный поиск для точек пересечения пути Чапаева и единственного болота. Такое решение работает за $O(\log C^2)$, где $\log C \approx 75$.

Для решения остальных двух подгрупп нужно заметить, что наиболее быстро Чапаев доберется до некоторой точки, если она будет лежать на пути, образованном начальным вектором пути, и удовлетворять закону Снеллиуса (отношение синусов углов наклона равна обратному отношению скоростей в соответствующих средах).

Для того, чтобы найти нужный вектор, можно ориентировать плоскость так, чтобы позиция Чапаева соответствовала точке $(0, 0)$, а позиция Зебрино — точке $(L, 0)$, где L — расстояние между соответствующими точками. После чего можно, используя бинарный поиск в интервале $[-\frac{\pi}{2}, \frac{\pi}{2}]$, найти искомый вектор, проверяя, окажется Чапаев «выше» точки $(L, 0)$ или нет.

Для полного решения нужно заметить, что последовательность пересечений болот всегда одинаковая, поэтому, найдя нужный порядок один раз с помощью сортировки, можно за $O(1)$ узнавать, с каким болотом будет следующее пересечение.

Итоговое время работы $O(NlogN + N \log C)$.