Задача A. Клоны

Автор задачи и разработчик: Егор Юлин

Заметим, что Морти i может иметь номер, равный i или $n-i+1$. Симметрично те же номера может иметь Морти с изначальной позицией в ряду $n-i+1$. Поэтому задача сводится к тому, чтобы проверить, что в каждой такой паре оба Морти нашлись (отдельно учесть случай нечетного n, где есть только один центральный без пары).

Тогда будем решать задачу следующим образом:

1. Получаем номер очередного Морти, обозначим его номер за i.
2. Если Морти с таким номером уже есть, то перейдем к следующему шагу, иначе добавим в нумерацию Морти с номером i.
3. Если после шага 2 у нас так же был Морти с номером $n-i+1$, то ответ «`NO`», так как не существует подходящей нумерации для текущего Морти (оба из пары уже найдены). Иначе добавим в нумерацию Морти, у которого теперь номер $n-i+1$.

Если после всех этих действий у нас получилась расстановка, в которой находятся все Морти с номерами от 1 до n, где каждое число встречается ровно один раз, то ответ «`YES`», иначе ответ «`NO`».

Задача B. Джеррикки и строка

Автор задачи и разработчик: Даниил Голов

Заметим, что в данной игре присутствует стратегия повтора. То есть, если один игрок как-то сходил, то следующим ходом другой игрок, если он вообще может сделать ход, может набрать больше очков, чем первый. Действительно, пусть игрок 1 поменял все буквы c_1 на c_2 и получил k очков. Значит, букв c_2 в строке стало k. Но тогда другой игрок может поменять любую другую оставшуюся букву на c_2 (или наоборот) и получить как минимум $k+1$ очков. Таким образом, оптимальной стратегией является игра в «повторение».

Теперь посмотрим, сколько ходов будет продолжаться игра. Каждый ход уменьшает количество уникальных букв в строке ровно на 1, при этом добавить в строку новые буквы никак нельзя. Тогда игра будет продолжаться $s-1$ ход, где s — количество уникальных букв в строке. Если $s-1$ четно, то выигрывает второй, так как всегда повторяет за первым, а значит, каждым ходом набирает больше очков. Если нечетно, то выигрывает первый, делая любой первый ход, а затем повторяя за вторым.

Задача C. Юнити

Автор задачи: Даниил Орешников, разработчик: Константин Бац

Подойдем к задаче с конца: у нас есть два стека, и мы хотим первой изъять единицу. Для этого мы должны перемещать элементы между стеками, пока 1 не окажется наверху одного из них. Понятно, что для этого необходимо вынуть из первого стека a все элементы выше 1 и переместить их во второй стек, после чего изъять единицу.

Заметим, что при этом если положить стеки a и b вершинами друг к другу, то относительный порядок элементов в них будет оставаться неизменным, пока мы просто перемещаем элементы между ними. Поэтому для повторения описанной выше операции для всех элементов от 2 до n достаточно поддерживать несколько величин:

- текущий i, который должен быть следующим вынут из стека;
- m — позицию «разделения» последовательности (количество элементов в стеке a).

Для каждого i от 1 до n, таким образом, надо просто определить его положение (первый или второй стек) и позицию в нем, после чего вынуть все элементы этого стека выше i, обновить ответ и обновить m. При обновлении m надо аккуратно учитывать, в каком из стеков находилось i. А чтобы находить позицию какого-то числа в его стеке, достаточно найти его позицию во всей последовательности, после чего найти его разность с m.

Находить позицию числа в последовательности — то же самое, что и находить число элементов, которые все еще лежат в последовательности левее него, что можно делать с помощью дерева

отрезков или дерева Фенвика с обновлением в точке и суммой на префиксе (поставим 0 или 1 в зависимости от того, был элемент удален или еще нет, и будем искать сумму таких флагов до исходной позиции интересующего нас числа). Суммарное время работы решения — $\mathcal{O}(n \log n)$.

Задача D. Деревянный Морти

Задача состоит в проверке, имеет ли первое дерево подграф, изоморфный второму дереву. Давайте попробуем решить задачу с любой асимптотикой, а затем уберем все лишнее из решения.

Сделаем первое дерево корневым и решим задачу с помощью динамического программирования на его поддеревьях. Пусть $\texttt{dp}[u][x \to v]$, где u — вершина первого дерева, а $x \to v$ — ориентированное ребро второго дерева, будет означать, возможно ли сопоставить вершины u и v, при этом

- сопоставляя поддерево u с поддеревом v;
- при условии, что во втором дереве родитель v — это x.

Иными словами, мы сопоставляем ребра $p(u) \to u$ и $x \to v$, а также поддеревья u и v.

Для подсчета такой динамики нам нужны значения всех $\texttt{dp}[u'][v \to v']$, где u' — сын u, а v' — сын v. Сам пересчет является проверкой, есть ли паросочетание в двудольном графе, которое покрывает левую часть — действительно, поддеревья каких-то детей u надо «покрыть» поддеревьями детей v, а всех остальных — добавить отдельно.

К сожалению, такая динамика работает очень медленно и не дает ответа на задачу. Давайте исправим обе эти проблемы сразу. Посмотрим на пересчет всех состояний динамики вида $\texttt{dp}[u][x \to v]$ для разных x. Для их пересчета мы ищем совпадение для графов с одинаковой правой частью (все дети u) и левой частью, которая отличается одной вершиной (все соседи v, кроме x).

Вместо этого давайте сразу запустим поиск паросочетания на графе со всеми интересующими нам вершинами, то есть только один раз для пары вершин u и v. Тогда:

- если мы нашли паросочетание, которое покрывает левую часть, то ответ для всех x будет `true`;
- если мы не смогли покрыть хотя бы две вершины, то значение всех $\texttt{dp}[u][x \to v]$ — это `false`;
- а если мы смогли покрыть все вершины, кроме одной x', то для всех остальных x ответ `true`, а для x' мы запустим один `dfs` от нее, ищущий дополняющую чередующуюся цепь, и определим ответ для $\texttt{dp}[u][x' \to v]$.

Решение заканчивается здесь, для ответа остается только рассмотреть какой-то лист в качестве x и найти, есть ли соответствующий `dp`, равный `true`.

Остается только оценить его время выполнения — мы запускаем один алгоритм поиска паросочетания для каждой пары вершин (u, v). Давайте посчитаем максимально возможное количество ребер во всех полученных графах, то есть $\sum\limits_{u,v} \deg(u) \cdot \deg(v) = \sum\limits_{u} \deg(u) \cdot 2 \cdot m = 4nm$.

Если честно оценить алгоритм Куна, то мы получим асимптотику всего решения $\mathcal{O}(nm^2)$, однако на конкретных графах алгоритм Куна на самом деле работает гораздо быстрее, поэтому такое решение уже проходит при достаточно аккуратной реализации. Если вы действительно хотите написать асимптотически более быстрое решение, напишите алгоритм Диница, чтобы получить решение за $\mathcal{O}(nm\sqrt{m})$.

Задача E. Риканутая перестановка

Автор задачи: Полина Шайдурова, разработчики: Полина Шайдурова и Даниил Орешников

Давайте заметим, что при циклическом сдвиге на 1 количество инверсий в перестановке меняется достаточно понятным образом. Пусть первым стоял элемент x. Тогда:

- элементы перестановки со второго по n-й останутся в том же относительном порядке, и количество инверсий между ними не изменится;
- ровно $x-1$ элемент перестановки меньше x, и все они стояли правее него, но после циклического сдвига будут стоять левее — число инверсий уменьшилось на $x - 1$;

- ровно $n - x$ элементов перестановки больше x, и все они стояли правее него, но после циклического сдвига будут стоять левее — число инверсий увеличилось на $n - x$.

Таким образом, при перемещении элемента x из начала перестановки в конец число инверсий изменяется на $n-2x+1$. Заметим, что, пройдя так целый круг, число инверсий вернется к исходному значению. А значит, чтобы найти минимальное возможное число инверсий, надо найти, на сколько максимум оно может уменьшиться за первые n циклических сдвигов.

Для этого просто выпишем новый массив b, в котором $b_i = n - 2a_i + 1$, посчитаем на нем префиксные суммы (суммарное изменение числа инверсий за первые сколько-то сдвигов) и найдем среди них минимальную. Позиция, на которой префиксные суммы достигают минимума, и будет ответом на задачу. Время работы решения — $\mathcal{O}(n)$.

Задача F. Загадка Прайма

Автор задачи и разработчик: Герман Андосов

Заметим, что если $n \equiv 2 \mod 3$, то мы можем узнать $(n-1)$-е число последовательности и умножить его на 2. Аналогично, если $n \equiv 0 \mod 3$, то мы можем узнать $(n-2)$-е число последовательности и умножить его на 3. Поэтому, давайте сделаем $n = \lceil \frac{n}{3} \rceil$ и будем далее рассматривать последовательность $1, 4, 5, 6, 7, 9, \ldots$.

Утверждается, что все числа в последовательности имеют вид $2^x \cdot 3^y \cdot p$, где p — натуральное число, не делящееся на 2 и на 3, а $(x + y)$ — четно. Это легко доказать по индукции.

Теперь заметим, что наша последовательность монотонна (в нашем случае строго возрастает), поэтому мы можем применить бинарный поиск. Для числа mid мы должны узнать, сколько чисел в последовательности стоят до него. Переберем значения x и y, и убедимся в том, что $(x+y)$ четно. Теперь ответим на вопрос — сколько чисел вида $2^x \cdot 3^y \cdot a$ меньше или равны x? Оказывается, их

$$T(x, y) = \left\lfloor \frac{mid}{2^x \cdot 3^y} \right\rfloor.$$

Теперь нам нужно из этого количества вычесть все числа, у которых a делится на 2 или на 3. Делается это по формуле включений-исключений:

$$T(x, y) \leftarrow T(x, y) - \left(\left\lfloor \frac{T(x, y)}{2} \right\rfloor + \left\lfloor \frac{T(x, y)}{3} \right\rfloor - \left\lfloor \frac{T(x, y)}{6} \right\rfloor \right).$$

.

Просуммировав значения $T(x, y)$ для всех чисел (x, y), мы получим искомую величину cnt (сколько чисел в последовательности стоят до числа mid). Если $cnt < n$, то сдвинем левую границу бинпоиска, иначе правую. Когда бинпоиск отработает, его правая граница и будет ответом на задачу. Не забываем, что иногда ответ надо будет домножить на 2 или на 3.

Задача G. Возвращение Злого Морти

Автор задачи: Даниил Орешников, разработчики: Даниил Орешников и Егор Юлин

Заметим следующий факт: если мы скажем, что отрезок $[a, b]$ «меньше» отрезка $[c, d]$ при $b < c$, то получится отношение частичного порядка, а ответом будет являться разбиение всех отрезков на минимальное количество возрастающих последовательностей. Это может быть подсказкой к решению: воспользуемся идеей из алгоритма поиска НВП за $\mathcal{O}(n \log n)$.

Но здесь, вместо поиска НВП, будем также обрабатывать отрезки по очереди слева направо, но поддерживать текущее «оптимальное» разбиение отрезков на группы. Под оптимальностью в данном случае будем понимать

- минимальное число групп;
- при фиксированном числе групп — лексикографически минимальную последовательность правых концов последних отрезков в каждой группе.

Аккуратным рассмотрением двух вариантов разбиения на группы можно показать, что добавление нового отрезка в более «оптимальное» разбиение приводит к не менее оптимальному новому разбиению, значит поддержание такого разбиения приведет к ответу, соответствующему минимальному числу групп в разбиении.

Осталось только научиться поддерживать это оптимальное разбиение. Отсортируем все отрезки по их левой границе и будем поддерживать правые концы всех групп в дереве поиска. Тогда, чтобы добавить очередной отрезок $[l, r]$, достаточно

1. найти максимальный конец группы r_i, который меньше l;
2. добавить отрезок $[l, r]$ в эту группу;
3. удалить из дерева поиска r_i и добавить вместо него r.

Действительно, добавление одного отрезка приводит к изменению правого конца ровно одной группы (либо к добавлению новой группы, если все $r_i \geqslant l$). Чтобы добиться лексикографически минимальной последовательности правых концов, следует заменить максимальный возможный r_i. Таким образом, все решение работает за $\mathcal{O}(n \log n)$.

Задача H. Тоннель

Автор задачи и разработчик: Даниил Голов

Заметим, что так как обгон в тоннеле запрещен, единственное, что могло изменить порядок выезда машин из тоннеля относительно порядка въезда — это разворот одной машины. Если разворот располагается ближе ко въезду, то развернувшаяся машина может поменять свое место в порядке на ближе к началу, если ближе к выезду — то сдвинуться в порядке ближе к концу.

Сделаем из последовательности a перестановку такого же формата, как последовательность c. Для этого можно, например, сделать из a последовательность пар $(a_i,\ i + 1)$ и отсортировать эту последовательность по первому элементу, а затем взять все вторые элементы. Будем называть такую получившуюся из a перестановку d.

Посмотрим на изменения в c относительно d. Если перестановки равны, то возможны несколько вариантов расположения разворота: разворот ближе к началу и развернулась машина, въехавшая первой; разворот ближе к концу и развернулась машина, въехавшая последней; разворот ровно посередине и развернулась любая машина. То есть в таком случае ответ «`impossible`». В случае, если две машины поменялись местами, тоже возможны несколько вариантов: либо та, которая переместилась в порядке вперед, развернулась, и тогда разворот ближе к началу; либо та, которая переместилась в порядке назад, развернулась, и разворот ближе к концу. Соответственно, в этом случае тоже «`impossible`».

В любом другом случае, так как мы знаем, что развернулась ровно одна машина, соответственно, одна машина переместится на несколько позиций вперед или назад, а остальные, которых она обогнала и от которых отстала, переместятся ровно на одну позицию назад или вперед. Таким образом, можно просто посмотреть на позицию машины, которая сдвинулась более, чем на один индекс в перестановке. Если она переместилась по порядку вперед, значит разворот был в начале, если назад — то в конце. Время работы решения — $\mathcal{O}(n \log n)$.

Задача I. Погоня за Риком Праймом

Автор задачи и разработчик: Даниил Орешников

Для начала перейдем в другую систему координат: работать с суммой по максимумам из двух величин неудобно. Множество точек, для которых $\max(|x-x_i|, |y-y_i|) = d$ — это квадрат со стороной $2d$ с центром в (x_i, y_i). Если повернуть плоскость на $45°$ и сжать координаты в $\sqrt{2}$ раз, получится «ромб» с тем же центром, но задаваемый равенством $|x - x_i| + |y - y_i| = d$. Таким образом, мы свели задачу к тому, чтобы найти такую (x, y), для которой в новых координатах

$$\sum_{i=1}^{n} p_i \cdot (|x - x_i| + |y - y_i|)$$

минимальна (разумеется, сжимать координаты при этом не обязательно, можно оставить все вычисления в целых числах).

А эту сумму уже можно разбить на две независимые суммы по x и по y, после чего выбирать x и y независимо. Разберем, как выбрать оптимальный x. Перейдем в новые координаты, сделав замену $x' \leftarrow x + y$, $y' \leftarrow x - y$. После чего, чтобы минимизировать $\sum p_i|x' - x'_i|$, заметим, что нам надо найти «взвешенную медиану» множества x_i, то есть точку, слева и справа от которой сумма p_i не превосходит половины всей суммы (иначе можно сдвинуть x' в сторону большей суммы p_i и уменьшить искомую сумму). А это можно сделать, просто отсортировав все точки по x'_i и пройдясь линейным проходом, считая сумму p_i.

Таким образом, найдем оптимальный x', после чего тем же образом найдем оптимальный y'. Чтобы вернуться в исходные координаты, достаточно взять $x = \frac{x'+y'}{2}$ и $y' = \frac{x'-y'}{2}$, но для вывода даже не надо делить их на 2. Время работы решения — $\mathcal{O}(n \log n)$.

Задача J. Гараж

Автор задачи и разработчик: Павел Скобелин

Заметим, что можно сразу избавиться от паролей, которые являются подстроками других строк. Для этого переберем все возможные пары строк, и проверим, находится ли строка s_i в строке s_j ($i \neq j$) с помощью любого подходящего алгоритма (к примеру, алгоритм КМП или хеши). Также нужно быть аккуратными с одинаковыми строками.

Теперь давайте построим ориентированный граф на n вершинах. Длина ребра (i, j) будет равна минимальному количеству символов, которые необходимо дописать в конец s_i, чтобы полученная строка содержала как подстроку строку s_j (менее формально, нас интересует максимальное «наложение» s_j справа на s_i). Давайте научимся считать эту величину: так как никакие две строки теперь не содержатся друг в друге, найдем максимальный суффикс s_i, совпадающий с префиксом s_j такой же длины. Тогда несложно понять, что искомая величина — это разность длины s_j и длины максимального совпадающего суффикса s_i и префикса s_j. А максимальные совпадающие префикс и суффикс можно найти, взяв последнее значение префикс-функции от строки «s_j#s_i», или посчитав аналогичное значение с помощью хешей.

Теперь заметим, что ответ на задачу — это гамильтонов путь минимальной длины в приведенном выше графе, с учетом того, что стартовать i-й вершины «стоит» длину строки s_i. Это несложно показать: посмотрим на вхождения данных строк в ответ: они идут в каком-то порядке. Очевидно, что нет смысла включать одну строку в этот порядок более 1 раза, ведь на графе выполняется неравенство треугольника. А чтобы для данного порядка минимизировать ответ, необходимо максимизировать «наложение» соседних пар строк — именно это мы и максимизируем в искомом графе.

В итоге мы получаем решение за $\mathcal{O}(2^n \cdot n^2 + S \cdot n^2)$ на поиск гамильтонова пути минимальной стоимости, и удаление строк, входящих в другие.

Задача K. Борской Мой

Автор задачи и разработчик: Владимир Рябчун

Для решения данной задачи применим динамическое программирование по изломанному профилю.

Будем постепенно заполнять поле кораблями, двигаясь по столбцам слева направо. Чтобы поставить какой-то корабль, необходимо знать, сколько кораблей данного вида осталось. Кроме этого необходимо хранить иформацию о предыдущих размещенных кораблях — они могут пересекаться или касаться того, который мы хотим поставить.

Будем идти по полю слева направо и сверху вниз. У нас в каждый момент будет одна клетка, про которую мы определяем, что с ней произойдет. Есть четыре варианта:

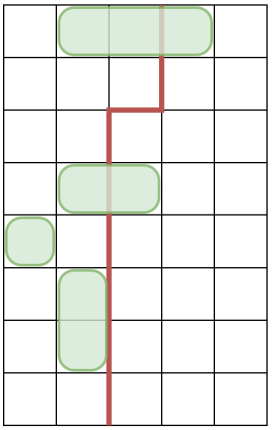
1. Оставить клетку путой. Для этого в ней не должно стоять символа ‘`x`’ и не должно быть уже поставленных кораблей, которые покрывают эту клетку.

2. Продолжить вправо корабль, размещенный до этого. Для этого нужно как-то помнить, а не «торчит» ли вперед какой-то старый корабль. О том, как это делать будет сказано далее.

3. Разместить в этой клетке корабль вертикально. Тогда он покроет эту клетку и несколько клеток ниже. Чтобы это было возможным, нужно проверить, не пересекаемся ли мы с уже размещенными кораблями и не пытаемся ли мы покрыть клетки, в которых записано '`o`'.

4. Разместить в этой клетке корабль горизонтально. Нужно опять же проверить, не мешает ли нам какой-то прошлый корабль и не нарушаем ли мы информацию про поле.

В итоге нужно помнить профиль кораблей — насколько они «торчат» за уже рассмотренные клетки. Предлагается для этого хранить битовую маску, однако наждой клетке будет отводиться два бита, кодирующие значения от 0 до 3. Значение 0 означает, что на границе нет клеток корабля, значение 1 означает, что на границе есть клетка, 2 — что корабль выпирает на одну клетку за границу, а 3 — что корабль выпирает на две клетки за границу.



В данном примере красной линией обозначен излом профиля, а зелеными фигурами — корабли. Состояние поля будет соответствовать паре чисел $(2, 0110202_4)$, где первое значение пары обозначает позицию излома профиля, а второе — описанную выше маску.

Итоговая динамика имеет вид $\texttt{dp}[c][\text{ship}_1][\text{ship}_2][\text{ship}_3][r][\text{mask}]$, где c — столбец, а r — излом.

Данная задача требовала довольно аккуратной реализации. Общее число масок для каждого профиля было не больше 2400, хотя числовые значения маски могли быть очень большими. Чтобы уменьшить потребление памяти, необходимо было сжать значения масок перед вычислением динамики. Кроме того, имело смысл делать динамику вперед и избавиться от первого измерения, храня только два последних столбца.

Для дальнейшего ускорения можно было для каждой клетки и для каждого из пяти возможных расположений кораблей на ней посчиать, а можно ли там разместить этот корабль. Это значительно ускоряет решение. Также можно для каждой маски и излома заранее посчитать, какой будет маска при каждом переходе. Последняя оптимизация, примененная автором задачи, заключалась в смене индексов динамики и порядке вычислений: возможность поставить корабль какого-то типа однозначно определяется уже при фиксированных (c, r, mask), поэтому стоило перебирать сначала их, а потом безусловно обновлять значения динамики в циклах по $\text{ship}_1, \text{ship}_2, \text{ship}_3$.

Итоговая асимптотика решения — $\mathcal{O}(wh \cdot s_1 s_2 s_3 \cdot \text{States})$, где $\text{States} \approx 2400$.

Задача L. Путешествие к примитиву

Автор задачи: Александр Гордеев, разработчик: Егор Юлин

Для удобства заменим слова на номер их следования во вводе (для этого просто пользуемся `unordered_map`. После этого построим ориентированный граф. У нас есть вершины (слова) и ребра между ними (замены).

Рассмотрим очевидные случаи, когда мы не можем уменьшить количество слов:

- У нас осталось ровно одно слово — очевидно, что уменьшить количество мы не можем.
- Оставшиеся слова находятся в разных компонентах слабой связности — в таком случае у нас нет ни одного способа заменить слова из разных компонент.

Введем более сильное утверждение: мы не сможем уменьшить количество слов, если все слова уже находятся в разных компонентах сильной связности, при этом ни одна не является предком любой другой. Действительно, в таком случае, очевидно, актуальных замен не осталось, а в противном случае — есть путь в графе, вдоль которого можно сделать замену.

Тогда будем действовать следующим образом: найдем все компоненты сильной связности, из которых не исходит ребер (стоки) — их количество и является ответом. Как мы показали выше, из ситуации, в которой в каждой из этих компонент осталось по одному слову, количество слов уменьшить не получится. Более того, это точная нижняя оценка на ответ, так как минимум по одному слову из каждой из этих компонент останется — их нельзя заменить ни на какое слово из других компонент. Получаем решение с временем работы $\mathcal{O}(n + m)$.