Разбор третьего этапа республиканской олимпиады по информатике, 2024

10 января 2024 г.

Задача 1-1. Замечательные частицы

Идея задачи: Егор Дубовик
Разбор: Илья Лазук

Ограничения: $n \leqslant 5$

Рассмотрим самый базовый случай. Минимальная строка, которая соответствует условиям задачи - aaabbbccc (3 вхождения a, 2 вхождения b, 1 вхождение c, 0 вхождений d). Данная строка состоит из 6 символов. Строку, которая состоит из меньше чем 6 символов и удовлетворяет условиям задачи, невозможно составить. Ответом на все случаи для $n \leqslant 5$ будет грустный смайлик.

Ограничения: $n \leqslant 10$

Можем перебирать все возможные значения для каждой буквы. Необходимо перебрать все возможные маски размера N, значениями в маске будут числа от 0 до 3. Число 0 в маске означает букву a, число b означает букву b. После чего мы легко можем проверить, что полученная в ходе перебора строка подходит под условия задачи. Из всех строк, полученных в ходе полного перебора, выбираем ту строку, которая отличается от исходной строки минимальным количеством букв.

Ограничения: $n(d) = n(c) = n(b)$

Вся строка состоит из букв a. Нам лишь необходимо совершить минимальное количество действий, чтобы количество букв, которые отсутствуют в строке, отличалось. К примеру, мы можем заменить любые две буквы a на букву b, и другую любую букву a на букву c. Итого имеем разное количество каждой буквы в строке за минимальное количество действий.

Ограничения: $n(d) = n(c)$

Решение схоже с предыдущим решением. Отличие заключается в том, что нам нужно проверить равенство количеств вхождений двух оставшихся букв и аккуратно обработать эти случаи.

Ограничения: $n(d) = 0$

Решение схоже с предыдущим решением. Отличие заключается в том, что нам нужно проверить равенство количеств вхождений трех оставшихся букв и аккуратно учесть эти случаи.

Полное решение

Давайте рассмотрим разные случаи.

- Количество вхождений всех букв совпадает. В данном случае мы можем получить строку, которая удовлетворяет условиям задачи, всего за три действия. К примеру, максимально увеличить количество одной буквы за счет других букв. $(3333) \leftarrow (4233) \leftarrow (5133) \leftarrow (6123)$
- Количество вхождений трех букв совпадает. В данном случае мы также стараемся увеличить одно из чисел. В случаях, когда четвертое число значительно отличается (абсолютная разница больше 1) от трех одинаковых чисел, мы можем за одну операцию привести последовательность к виду, который удовлетворяет условиям задачи. Иначе мы сможем сделать это за 2 операции.
- Количество вхождений двух букв совпадает. Если два других оставшихся числа значительно отличаются (абсолютная разница больше 1) от наших равных чисел, мы можем за одну операцию изменить последовательность. Иначе, нам понадобится до 4 операций.

Если внимательно изучить разбор случаев, то можно заметить, что можно получить ответ не более чем за 4 операции. Посчитаем все маски вида $[a, b, c, d]$, где a — количество букв a, которое нужно поставить на место других букв в строке. Если значение отрицательное, то нужно убрать именно столько букв a из строки (заменить их на другие буквы). Аналогично и для других букв. Напишем цикл, который перебирает все возможные вариации данной маски (цикл 4-ой вложенности, значения маски от -4 до 4 включительно). Критерий валидной маски - сумма $a + b + c + d = 0$ и количество вхождений всех букв после применения маски различно, Найдем в полном переборе минимальную подходящую маску. Восстановление ответа: если текущее значение i в маске отрицательно, то заменяем букву, которая соответствует элементу этой маски на любую букву, значение маски которой положительно. Вносим соответствующие изменения в маску. Поскольку маска имеет нулевую сумму, для отрицательного значения в маске всегда найдется положительное.

Задача 1-2. Редкий сорт

Идея задачи: Антон Керножицкий
Разбор: Кирилл Байдаков

Ограничения: $n \leqslant 3000$

Подвешиваем дерево поочередно за каждую вершину, назовем ее a. Тогда за $O(n)$ попробуем проверить для всех остальных вершин, содержат ли пути в качестве подпоследовательности нашу строку t. Чтобы это проверить будем обходить дерево с помощью поиска в глубину, параллельно жадно поддерживая максимальный префикс, который мы можем набрать по пути. Решение за $O(n^2)$.

Ограничения: $|t| \leqslant 3$

Если $|t| \leqslant 2$, то нам просто нужно проверить, что существуют такие символы в строке s, потому что мы в любом случае можем построить путь между ними.

Если $|t| = 3$, то при поиске в глубину нам нужно проверить что либо:

1. Есть такая вершина значение в которой равно второму символу строки t и существуют два **различных** потомка, поддеревья которых содержат первый и последний символы строки t.

2. Есть такая вершина, которая в своей вершине содержит второй символ строки t, на пути от корня до которой есть первый(либо последний) символ строки и в ее поддереве (не включая ее саму) есть последний(или первый соответственно) символ.

Полное решение

Воспользуемся методов динамического программирования по поддеревьям. Для каждой вершины v нужно насчитать максимальный префикс строки t, который можно набрать, если начать из какой-то вершины поддерева v и закончить в вершине v, будем хранить это значение в $dp_{v,0}$, и максимальный суффикс, который можно набрать если начать в вершине v и закончить в какой-то вершине поддерева v, это значение храним в $dp_{v,1}$.

Базовый случай ДП:

$dp_{v,0} = 1$, если $s[v] = t[0]$, иначе 0.
$dp_{v,1} = 1$, если $s[v] = t.back()$, иначе 0.

Пересчет ДП:

$dp_{v,0} = max(\sum_{u \in subtree[v]} dp_{u,0} + (1, \text{если } s[v] = t[dp_{u,0}], \text{иначе } 0), dp_{v,0})$

$dp_{v,1} = max(\sum_{u \in subtree[v]} dp_{u,1} + (1, \text{если } s[v] = t[t.size() - dp_{u,1} - 1], \text{иначе } 0), dp_{v,1})$

Тогда для того, чтобы пересчитать ответ, нам нужно проверить, что либо какое-то из состояний ДП полностью покрывает строку t, либо существует такая вершина, у которой существует потомки $u1$ и $u2$, такие что объединение $dp_{u1,0}$, $s[v]$, $dp_{u2,1}$ дает строку t.

Итоговая сложность: $O(n)$.

Задача 1-3. Сборный вопрос

Идея задачи: Антон Керножицкий, Александр Керножицкий
Разбор: Кирилл Байдаков

Ограничения: $n \leqslant 2000$

Рассмотрим все возможные пары a и b и разобьем их на две группы:

1. $b \geqslant a$: Переберм для каждого b все a, которые меньше b, найдем их НОД и, очевидно, что при такой паре a и b число $k = \frac{b}{gcd(a,b)}$. Тогда в $sumb[i]$ находится суммарный вклад в ответ, при $b = i$ и $a \leqslant b$.

2. $a > b$: Аналогично с первым случаем переберем все такие b, что $b < a$. И также в $suma[i]$ посчитаем суммарный вклад в ответ, при $a = i$ и $b < a$.

Если посчитать префиксные суммы по массивам $suma$ и $sumb$. Тогда ответом на каждый тест будет: $\sum_{i=1}^{n} suma[i] + sumb[i]$.

Полное решение

Заметим, что все возможные k, на которые нужно домножить a представляются в виде $\frac{b}{u}$, где u - все делители числа b. Тогда модифицируем подсчет массивов $suma$ и $sumb$.

1. $b \geqslant a$: Пройдемся по всем делителям числа b, потому что всевозможные $a < b$ могут иметь НОД с b только равный какому-то делителю b. Для конкретного делителя u: все рассматриваемые a должны быть вида $k * u$, где $k \in [1..\frac{b}{u}]$, но также k должно быть взаимно простым с числом $\frac{b}{u}$, потому что тогда $gcd(a, b)$ станет больше u, а мы рассматриваем конкретный случай с $gcd(a, b) = u$. Тогда по сути количество чисел $k \in [1..\frac{b}{u}]$ и $gcd(k, \frac{b}{u})$ в точности равно функции Эйлера от числа $\frac{b}{u}$.

   Тогда вклад всех чисел a, у которых $gcd(a, b) = u$ равен: $\frac{b}{u} * \varphi(n)$.

2. $a > b$: Пройдемся по всем делителям числа a, потому что всевозможные $b < a$ могут иметь НОД с a только равный какому-то делителю a. Для конкретного делителя u: все рассматриваемые b должны быть вида $k * u$, где $k \in [1..\frac{b}{u})$. Тогда суммарный вклад по всем таким парам при фиксированном a равен сумме таких k, что $gcd(k, b) = 1$. Отсюда понятно, что достаточно просто найти все такие k, что у них нет общих делителей с числом b.

   Можно заметить, что для $n \leqslant 5 \cdot 10^5$ у каждого числа не более 6 различных простых делителей. Тогда мы можем с помощью масок и метода включения/исключения посчитать сумму по всем нам валидным k.

Нахождения функции Эйлера, всех делителей и, отдельно, различных простых делителей для n затратит $O(n \cdot log(n))$ времени. Поиск вклада по всем парам первого типа займет $O(n \cdot log(n))$. Поиск вклада по всем парам второго типа займет $O(n \cdot log(n) \cdot 2^6)$.

Итоговая сложность: $O(n \cdot log(n) \cdot 2^{log(log(n))})$.

Задача 1-4. Пасьянс «Паук» 2.0

Идея задачи: Егор Анисевич, Андрей Мищенко
Разбор: Антон Керножицкий

Динамическое программирование за $O(n^2m)$ может быть написано довольно несложно.

Сортируем отрезки по неубыванию левой границы. Ответ, лежащий внутри ДП, всегда будет иметь следующий вид: сначала сколько-то клеток с двумя пауками (далее двойка), потом сколько-то клеток с одним пауком (далее единица), и остальные клетки без пауков. Это нетрудно показать. В самом деле, пусть существует какая-то клетка с 0 или 1 пауками, справа от которой находится большее число пауков. Это возможно в двух случаях:

- В эту клетку поставили одного паука, не используя запутанные отрезки. Тогда эту операцию можно было совершить раньше или позже других операций, получив результат, описанный выше. При этом ответ бы не изменился, поскольку он не меняется от порядка операций.
- В эту клетку поставили одного паука, используя запутанные отрезки. Тогда можно до этого отрезка расставить недостающих пауков, поскольку отрезки отсортированы, и никакой другой отрезок уже не сможет доставить пауков в эти клетки.

Переходы в таком ДП довольно прямолинейны и оставляются как упражнение читателю.

Теперь соптимизируем это ДП до $O(n^2)$. Здесь используется важный факт: в любом положении, которое характеризуется самой правой двойкой

(пусть это i) и самой правой единицей, оптимально использовать такой отрезок (l, r), что $l \leqslant i + 1$, и среди всех таких отрезков его правая граница максимальна. Если такого отрезка нет, это означает, что $i + 1$-ую клетку невозможно покрыть отрезком, а значит, ее надо покрыть не используя отрезки. Таким образом, получаем три перехода:

- Добавить отрезок
- Заменить самую левую единицу на двойку
- Заменить самый левый ноль на единицу

Для решения за $O(m^2)$ достаточно применить сжатие координат. Оно может быть выполнено различными способами, но необходимо аккуратно учитывать два последних перехода (от этого константа в сложности может стать достаточно большой).

Используя схожее утверждение, что и факт про отрезки, можно получить решение жадным алгоритмом за $O(mlogm)$. Достаточно хранить самую правую двойку и аналогично брать отрезок с наибольшей правой границей.

Итоговая сложность: $O(m \cdot log(m))$.

Задача 2-1. Неполадки в полете

Идея задачи: Андрей Мищенко
Разбор: Владислав Вишневский

Для начала давайте заметим, что n всегда должно делиться на m и каждое число от 1 до m должно встречаться в массиве a ровно $\frac{n}{m}$ раз для успешной настройки маршрута.

Далее нам нужно проверить можно ли разбить массив a на $\frac{n}{m}$ непересекающихся подпоследовательностей с числами $b_1, b_2, \ldots, b_m$. Такую проверку мы можем выполнить сделав $\frac{n}{m}$ итераций следующим жадным алгоритмом:

1. определим для каждого i от 1 до m число p_i — минимальную позицию в массиве a, где находится число b_i и она не была использована на предыдущих итерациях.
2. все элементы p_i должны следовать в отсортированном порядке для успешного завершения шага. Поскольку, если найдется такое i для которого $p_i > p_{i+1}$ это будет обозначать, что элемент p_{i+1} не может быть использована в будущем т.к. не существует числа b_i в массиве a с позицией меньше чем p_i.
3. пометим все элементы с индексами $p_1, p_2, \ldots, p_m$ как использованные.

Данный алгоритм можно реализовать за сложность $O(n)$ используя вспомогательный массив хранящий для каждого числа i от 1 до m позиции элементов массива a со значением i в отсортированном порядке.

Итоговая сложность: $O(n)$.

Задача 2-2. Показательный отчет

Идея задачи: Антон Керножицкий
Разбор: Кирилл Байдаков

Ограничения: $n \leqslant 5000$

Посчитаем для всех $O(n^2)$ подотрезков минимальные значения на них. Во время подсчета для всех возможных длин подотрезков будем запоминать максимальное значение минимума на всех подотрезков такой длины.

Ограничения: $a_i \leqslant 10$

Переберем всевозможные значения минимума (в порядке убывания) на отрезках. При фиксированном значении минимума оставим в массиве только те элементы, которые больше либо равны перебираемому минимуму.

За $O(n)$ найдем отрезок максимальной длины с фиксированным минимумом. Пусть это будет, например, len. Тогда попробуем обновить ответ для $k \in [1..len]$. То есть смотрим для каждого такого k: если мы на прошлых операциях не нашли отрезка с большим минимумом, то обновим значение ответа для этого k на текущий перебираемый минимум.

Полное решение

Заведем массив с элементами в виде пар: (значение элемента, индекс в исходном массиве). Отсортируем этот массив по невозрастанию значений, а при равенстве значений по невозрастанию индексов.

Далее заведем массив, в котором изначально все элементы равны 0. Будем поочередно (в порядке невозрастания элементов) в нашем массиве пар "активировать"элементы на соответствующих позициях, то есть ставить 1 на соответствующую позицию. Тогда при каждой такой активации какого-то элемента будем находить максимальный по длине подотрезок из единиц.

А это уже бяан. Для решения воспользуемся ДО, в вершинах которого нужно поддерживать максимальный по длине отрезок из единиц, максимальный суффикс из единиц, максимальный префикс из единиц и длина отрезка (для удобства). Тогда в корневой вершине ДО всегда будет находиться максимальный отрезок из единиц на всем массиве

Теперь когда мы умеем находить максимальный отрезок из единиц при известном нам минимуме, то найти ответ для всех k не составит труда. Для этого при нахождении максимального отрезка из единиц, нужно еще и знать каким был максимальный отрезок по длине при предыдущем (не меньшем) минимуме. Тогда пусть при предыдущем минимуме максимальная длина равна $prefk$, для текущего - $currk$. Тогда, очевидно, текущий перебираемый минимум будет лучший ответом для всех $k \in [prefk+1...currk]$, потому что значения минимумом по ходу алгоритма невозрастает.

Итоговая сложность: $O(n \cdot log(n))$.

Задача 2-3. Заведомо сложная реклама

Идея задачи: Андрей Мищенко
Разбор: Андрей Мищенко

Подсчитаем для всех $O(n^2)$ подотрезков среднее значение. Теперь наша задача – найти k-е число за линейное время. Можно воспользоваться встроенным методом в **C++ nth_element()**.

Ограничения: $0 \leqslant a_i \leqslant 1$

Среднее значение на подотрезке будет равно единице, только если этот подотрезок полностью состоит из единиц. То есть для решения этой группы достаточно подсчитать, сколько подотрезков полностью состоят из единиц. Это можно сделать за линейное время. Будем проходиться слева направо и поддерживать количество подряд идущих единиц на суффиксе и добавлять это к итоговому счетчику.

Ограничения: $a_i = i$

Среднее значение на подотрезке будет равно значению медианного числа. Это означает, что у нас существует n различных значений подотрезков. Давайте для каждого числа посчитаем, сколько существует подотрезков. Для числа a_i это будет $min(i, n - i + 1) + min(i, n - i)$.

Полное решение

Давайте научимся проверять, что набор чисел имеет среднее арифметическое хотя бы x. Изначально у нас неравенство: $\frac{b_1+b_2+\cdots+b_m}{m} \geqslant x$. Мы можем привести его к виду: $(b_1 - x) + (b_2 - x) + \cdots + (b_m - x) \geqslant 0$.

Давайте перебирать ответ бинарным поиском. Пусть $a'_i = a_i - x$. Нам нужно посчитать, сколько подотрезков со средним арифметическим меньшим, чем x. Это равносильно количеству подотрезков, у которых сумма a' отрицательная.

Посчитаем префиксную сумму для массива a'. Теперь нужно решить довольно известную задачу: посчитать количество пар $l \leqslant r$, таких, что $pref_r < pref_l$. Это можно сделать при помощи структур данных, например, заранее сжать значения сумм и воспользоваться деревом Фенвика. Также это можно было делать при помощи других структур данных, например, используя дерево отрезков.

Итоговая асимптотика: $O(n \cdot \log(n) \cdot \log(10^9))$.

Задача 2-4. Задача про кактусы

Идея задачи: Александр Керножицкий
Разбор: Антон Керножицкий

Тест №1

Этот тест представлял собой следующую конструкцию: необходимо расставить 32 коня на шахматной доске. Нетрудно видеть, что для этого необходимо поставить всех коней на поля одинакового цвета (так как конь при ходе всегда меняет цвет поля).

Тест №2

Этот тест предлагал расставить 10 фигур на доске 10×10. Эти фигуры ходили как ферзь и конь одновременно. Предполагаемое решение: перебор всех вариантов. Нетрудно видеть, как можно реализовать перебор со сложностью $O(n! \cdot n)$.

Интересная ссылка: https://oeis.org/A051223

Тесты №3, №4

В этих тестах предлагалось решить известную задачу для больших ограничений: расставить n ферзей на доске $n \times n$, где n равно 111 в третьем тесте и 2000 в четвертом.

Предполагаемое решение: у этой задачи есть много решений, одно из них — несколько упрощенная реализация генетического алгоритма.

Тесты №5, №6

В этих тестах предлагалось решить следующую задачу: дополнить латинский квадрат (12×12 и 26×26 в пятом и шестом тестах соответственно), если несколько чисел уже даны и полностью расставлены. Эта задача может быть решена довольно быстро. И генерация, и решение таких тестов может быть выполнено за полиномиальное время (нетрудно найти как минимум решения за $O(n^4)$ и $O(n^{3.5})$).

Рассмотрим теорему Холла. Ее формулировка гласит: в двудольном графе для любого натурального k любые k вершин одной из долей, где k не превышает числа вершин доли, связаны по крайней мере с k различными вершинами другой доли тогда и только тогда, когда граф разбивается на пары по первой доле.

Докажем теперь такое утверждение: в каждом r-регулярном двудольном графе для $r > 1$ существует совершенное паросочетание. Регулярным графом называется граф, степени вершин которого одинаковы, а совершенным паросочетанием называется паросочетание из всех вершин графа. Доказательство этого факта можно найти в статье.

Нашу таблицу можно представить в виде двудольного графа из $2n$ вершин. Более того, поскольку некоторые числа заполнены строго полностью (по n штук каждого), граф регулярен. Отсюда получаем, что в нем существует совершенное паросочетание. Если его удалить, снова останется регулярный граф, в котором снова будет существовать совершенное паросочетание. Отсюда следует, что для решения данных тестов можно использовать следующий алгоритм: искать наибольшее паросочетание, и ставить на это место первое число, которого еще нет, после чего повторять процесс, пока вся таблица не будет заполнена. Для генерации тестов можно использовать аналогичный алгоритм.

Тест №7

В этом тесте была дана таблица $1 \times n$, было всего два типа кактусов, и все зависимости имели $dy_i < 0$. Нетрудно видеть, что такая задача может быть решена динамическим программированием за $O(n^2 \cdot h)$.

Тест №8

В этом тесте не было загрязненных клеток, и выполнялось $|dx_i| \leqslant 5, |dy_i| \leqslant 10$. Можно заметить, что довольно хорошим решением может быть то, которое повторяется много раз на большой таблице. Авторский ответ выглядит так:

0 1 1 1 1 0 2 0 2 0 2 0 2 0 1 1 1 1 2 0 2
2 1 1 1 1 2 0 2 0 2 0 2 0 2 1 1 1 1 0 2 0
0 1 1 1 1 0 2 0 2 0 2 0 2 0 1 1 1 1 2 0 2
2 1 1 1 1 2 0 2 0 2 0 2 0 2 1 1 1 1 0 2 0

Предполагаемым решением так же может быть решение для тестов 9 и 10.

Тест №9, №10

Эти тесты предполагали использование различных оптимизационных алгоритмов, таких, как метод имитации отжига и генетический алгоритм. Приведем краткое описание авторского решения.

Воспользуемся следующим методом. Для каждого кактуса будем хранить его признаки: координаты x и y, его тип, а также стоимость.

Будем подбирать веса для этих признаков, используя метод имитации отжига. Затем будем добавлять кактусы в порядке невозрастания стоимостей.