Задача A. Две корзины

Автор задачи:	Владислав Макаров
Подготовка тестов и решений:	Владислав Макаров
Автор разбора:	Владислав Макаров

Ответ на задачу равен $n - 1$. Давайте разбираться, почему это так.

Ясно, что Боре всегда выгодно опустошать ту корзину, в которой больше яблок. Пусть Аня каким-то способом добилась того, что после хода Бори в корзине, которую он не тронул, ровно k яблок. Когда Аня может добиться какого-то «прогресса», то есть сделать так, что после следующего хода Бори в нетронутой корзине будет $k + 1$ яблоко (или больше)? Заметим, что Ане не важно, насколько медленно она «продвигается вперёд», если она всё-таки это делает, поскольку игра продолжается бесконечно, а каждое продвижение делает её ситуацию строго лучше. Поэтому, если считать, что каждый раз она добивается прогресса в ровно одно яблоко, то ответ не изменится.

Продвинуться вперёд она может в том и только в том случае, когда возможно доложить n яблок в корзины таким образом, что в каждой из них будет хотя бы $k + 1$ яблоко. То есть, когда $n \geqslant (k+1)+1 = k+2$ (в пустую корзину нужно доложить хотя бы $k+1$ яблоко, а в непустую — хотя бы одно). Таким образом, пока $k \leqslant n - 2$, Аня может добиться прогресса. Значит, ответ не меньше $(n - 2) + 1 = n - 1$. То же самое рассуждение показывает, что Аня не может добиться ответа n: для этого нужно доложить хотя бы $(n - 1) + 2 = n + 1$ яблоко, а она докладывает только n.

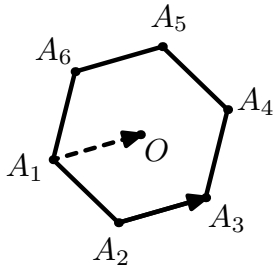
Задача B. Восстановление шестиугольника

Автор задачи:	Иван Казменко
Подготовка тестов и решений:	Иван Казменко
Автор разбора:	Иван Казменко

Правильный шестиугольник можно восстановить по любым трём вершинам, заданным в произвольном порядке. Или даже по двум, если известно, какие именно это вершины и в каком порядке они даны. Тем не менее, какие-то решения удобны, а другие состоят из большого количества случаев.

Одно из удобных решений — при первом запуске отметить две соседние вершины и центр шестиугольника. Выведем первую вершину (назовём её A_1). Посмотрим в цикле на все остальные вершины, найдём любую ближайшую к ней (назовём её A_2) и выведем её тоже. Наконец, посчитаем среднее арифметическое всех шести вершин по каждой координате и выведем полученную точку (назовём её O) — это будет центр шестиугольника.

При восстановлении будем складывать векторы, чтобы получать следующие вершины A_i: $A_i = A_{i-1} + \overline{A_{i-2}O}$. Действительно, сторона A_2A_3, например — это просто отрезок A_1O, отложенный из вершины A_2.



Задача C. Домашнее задание

Автор задачи: Александр Марков
Разработчик: Александр Марков
Автор разбора: Александр Марков

Во-первых, очевидно, что нужно решить задачу в отдельности для каждого s_i и просуммировать ответы.

Для этого рассмотрим подстроку t в s_i после добавления каким-то образом в s_i нужных символов. Какие-то символы получившейся подстроки остались из s_i, а какие-то были вставлены между ними или добавлены перед ними или после. В итоге символы, которые изначально были в s_i, образуют в этой подстроке t подпоследовательность. Следовательно, задача сводится к поиску наибольшей подстроки s_i, которая является подпоследовательностью в t. Эту задачу можно решить множеством способов.

На 20 баллов можно, например, перебрать за $O(2^{|t|})$ все подпоследовательности строки t и сохранить в `set`, а затем перебрать за $O(|s_i|^2)$ все подстроки s_i и для каждой проверить, есть ли такая подстрока во множестве, а из всех подходящих взять наибольшую.

На 60 баллов можно было перебрать все подстроки s_i и для каждой проверить методом двух указателей, является ли рассматриваемая подстрока подпоследовательностью в t. Для этого нужно, смещая каждый раз вправо указатель в подстроке s_i, сдвигать указатель в t, пока они не станут указывать на одинаковый символ. Если указатель в t дойдёт до конца строки, не найдя нужный символ, то нужной подпоследовательности в t нет.

Это решение несложно улучшается до решения на 100 баллов. Для этого заметим, что метод двух указателей может не только проверить, является ли фиксированная подстрока подпоследовательностью, но и найти самую большую такую подстроку с началом в данной позиции. Для этого, поставив указатель изначально в некоторый символ s_i (фиксированное начало подстроки), а указатель в t на начало, будем сдвигать указатель в s_i, пока указатель в t не дойдёт до конца строки. Так можно найти самую большую подстроку в s_i с началом в данной позиции, являющуюся подпоследовательностью t. Выполним это для всех позиций в s_i. Такое решение работает за $O(|s_i| \cdot |t|)$.

Также на 100 баллов можно посчитать динамику, похожую на динамику для подсчёта редакционного расстояния.

Задача D. Жизнь на Марсе

Автор задачи: Александр Марков
Разработчик: Александр Марков
Автор разбора: Александр Марков

На 40 баллов можно было просто построить полный граф из клеток поля и найти кратчайший путь в нём при помощи обычного алгоритма Дейкстры. Для перемещения из клетки (i_1, j_1) в клетку (i_2, j_2) потребуется $|i_2 - (i_1 + r_{i_1 j_1})| + |j_2 - (j_1 + c_{i_1 j_1})|$ единиц заряда — это и будет весом ребра между клетками (i_1, j_1) и (i_2, j_2). К сожалению, в этом графе $O(n^2 m^2)$ рёбер — слишком много при $n = m = 1000$.

Для полного решения на 100 баллов нужно придумать граф поинтереснее. Во-первых, по направлению вектора поля всегда можно переместиться, использовав 0 единиц заряда — добавим соответствующие рёбра веса 0. Также заметим, что, если можно добраться в какую-либо клетку (r, c), затратив d единиц заряда, то во все смежные по стороне клетки можно добраться за $d + 1$ единицу заряда — для этого необходимо изменить на 1 соответствующую координату в векторе мощности, который использовался для того, чтобы попасть в клетку (r, c). Следовательно, добавим ребро веса 1 из каждой клетки во все смежные с ней по стороне. Однако есть исключение — такое ребро нельзя добавить в стартовую вершину, ведь изначально нет вектора мощности, который использовался для

того, чтобы в неё попасть, и менять нечего. При этом такие рёбра служат для корректировки векторов мощности, и рёбра по сторонам из начальной клетки могут понадобиться для корректировки вектора мощности из какой-либо другой клетки. Для того, чтобы решить эту проблему, давайте просто разделим начальную вершину на две, то есть разрешим один раз зайти в начальную вершину повторно. Из первой копии мы начинаем путь, и из неё выходит только одно ребро по вектору поля, а у второй копии совпадает координата, но в неё мы как-то вернулись — значит, можно добавить не только ребро по вектору поля веса 0, но и рёбра веса 1 в клетки, смежные по стороне. Ответом на задачу будет кратчайший путь в получившемся графе — его можно найти при помощи того же алгоритма Дейкстры или при помощи 0-1 BFS. В нашем новом графе всего $O(nm)$ рёбер, поэтому такое решение будет работать достаточно быстро.

Задача E. А ну повтори!

Автор задачи:	Михаил Иванов
Подготовка тестов и решений:	Михаил Иванов
Автор разбора:	Михаил Иванов

Выделим на холсте 2^n непересесекающихся белых прямоугольников такого же размера, что и картина. Будем говорить, что прямоугольник готов на i шагов, если в нём i чёрных непростреленных клеток, которые должны быть и в картине чёрными, а остальные белые (и, очевидно, тоже непростреленные). Мы выиграем, если получим один прямоугольник, готовый на B. Его мы получим за B стадий, в течение i-й из которых мы получим 2^{n-i} непересекающихся прямоугольников, готовых на i. Сделать это из 2^{n-i+1} непересекающихся прямоугольников, готовых на $i-1$, очень просто: разбить их на пары, каждый день в одной из пар красить две чёрных клетки, а потом ночью Микола испортит выстрелом не более одного прямоугольника. За 2^{n-i} дней и ночей мы получим 2^{n-i+1} прямоугольников, максимум 2^{n-i} из которых будут испорчены, так что можно среди выживших выделить $2^{n-i+1} - 2^{n-i} = 2^{n-i}$, и все они будут готовы на i.

На покраску уйдёт $2^{B-1} + \ldots + 2^1 + 2^0 = 2^B - 1 \leqslant 8191$ дней, это и требовалось.

Задача F. Сумма расстояний

Автор задачи:	Владислав Макаров
Подготовка тестов и решений:	Владислав Макаров
Автор разбора:	Владислав Макаров

Ограничения в этой задаче могут немного сбить с толку: может создаться впечатление, что требуется придумать какое-то быстрое экспоненциальное решение. Тем не менее, жюри не знает ни одного экспоненциального решения, которое могло бы получить полный балл без очень долгого предподсчёта. Более того, эту задачу можно решить с гораздо бóльшими ограничениями.

Дальнейший план разбора такой: мы придумаем *какое-то* полиномиальное решение этой задачи, поймём, что на самом деле его хватает, чтобы получить 100 баллов, и вкратце обсудим, как решать эту задачу ещё быстрее, а также экспоненциальные решения, набирающие много баллов.

Сперва придумаем более простое выражение для величины, которую мы суммируем по деревьям. Пусть мы зафиксировали какое-то (помеченное) дерево с n вершинами. Посмотрим на какое-то его ребро e. Для скольки пар вершин это ребро лежит на пути между ними? Для $\ell(e) \cdot (n - \ell(e))$, где $\ell(e)$ — размер (в вершинах) любого из двух деревьев, на которые разобьётся исходное дерево при удалении ребра e (концы пути должны лежать в разных компонентах, здесь и далее под компонентами подразумеваются деревья, полученные из исходного удалением ребра e). Следовательно, искомая величина есть $\sum_{e \text{ — ребро дерева}} \ell(e) \cdot (n - \ell(e))$.

Отлично, теперь сделаем несколько наблюдений, которые подтолкнут нас к правильному решению. Первое наблюдение состоит в том, что вряд ли для ответа есть какая-то простая математическая формула. Интуитивно понятно, что расстояния между вершинами — достаточно сложная и «нелокальная» вещь, которую сложно отловить с помощью явных формул. Например, если вы слышали про коды Прюфера, то по коду Прюфера легко понять степени всех вершин дерева (степени вершин — это что-то очень локальное), но, непонятно, как искать диаметр дерева, не строя его явно и не делая что-то ещё более сложное.

Раз формулу мы придумать не можем, а перебирать всё — слишком долго, то задача почти гарантированно решается с помощью динамического программирования. Что не очень удивительно, так как большинство задач на деревья решаются какой-нибудь динамикой, но обычно у нас есть дерево, а тут его нет. Это не проблема — просто мы будем дерево одновременно строить и считать на нём динамику.

Вернёмся пока временно к случаю, когда нам дано какое-то дерево. Вспомним выражение для суммы расстояний: $\sum_e \ell(e) \cdot (n - \ell(e))$. Каждое слагаемое этого выражения очень слабо зависит от ребра: важно только $\ell(e)$. Представим, что мы удалили ребро e из дерева. Тогда оно разбилось на два. Чтобы посчитать ответ, нужно сложить сумму вкладов рёбер из одной компоненты, сумму вкладов рёбер из другой компоненты и вклад ребра e, который зависит только от размеров компонент. Более того, для получившихся компонент мы можем повторить то же самое рассуждение: рассмотреть любое ребро, учесть его вклад, удалить его, добавить вклады получившихся ещё меньших компонент, и так далее. Здесь я «замял» тонкий момент, который мы обсудим позже, когда дойдём до решения.

Может показаться, что мы просто придумали очень сложный способ посчитать сумму $n - 1$ чисел. Однако, у этого способа есть преимущество: каждый раз, когда мы рассматриваем какую-то компоненту и ребро в ней, нам нужно знать *не так уж много* про компоненты, образующиеся при удалении этого ребра: их размер и сумму вкладов рёбер из них. Зная эти величины, мы можем посчитать размер и сумму вкладов рёбер исходной компоненты. Это и есть основная идея динамического программирования: мы свели решение какой-то задачи к решению более маленьких задач такого же вида.

Итак, наша цель состоит в том, чтобы получить все возможные (помеченные) деревья на n вершинах склейкой n компонент из одной вершины, при этом ровно по одному разу. Более того, поскольку от каждой промежуточной компоненты нам нужно помнить лишь два числа, а не всю структуру дерева на ней, то выглядит очень правдоподобным, что мы сможем динамикой такого вида вычислить сумму расстояний по всем помеченным деревьям.

Нам хочется посчитать динамику «число возможных компонент с vers вершинами и суммой вкладов рёбер, равной sum», что бы эти слова не значили. Осталось разобраться с деталями. Помимо мелких технических подробностей, здесь возникает тонкий момент, о котором я предупреждал раньше: мы должны суммировать вклады рёбер в сумму расстояний в **исходном** дереве, а не в рассматриваемой компоненте. То есть $\ell(e) \cdot (n - \ell(e))$, а не $\ell_{\text{компонента}}(e) \cdot (\text{vers} - \ell_{\text{компонента}}(e))$, где vers — число вершин в рассматриваемой компоненте, а $\ell_{\text{компонента}}(e)$ — размер одной из подкомпонент, на которые компонента разбивается при удалении e. Даже $\ell_{\text{компонента}}(e) \cdot (n - \ell_{\text{компонента}}(e))$ — не совсем то, что нам нужно, если только не оказалось так, что $\ell_{\text{компонента}}(e) = \ell(e)$.

В идеале хочется, чтобы всегда выполнялось условие $\ell_{\text{компонента}}(e) = \ell(e)$. То есть каждый раз, когда мы «склеиваем» две какие-то компоненты с помощью ребра e, одна из них должна быть одной из компонент **исходного** дерева, получаемых удалением из него ребра e.

Легко понять, что не для всех порядков склеивания это условие будет выполняться, например «бамбук» можно склеить из одной вершины справа, большой середины и одной вершины слева. Тогда середина не будет удовлетворять нужному условию.

Есть такой естественный порядок склейки: подвесить исходное дерево за вершину с номером 1, а потом поочерёдно приклеивать поддеревья детей каждой вершины к ней. Например, если у нас есть вершина v с тремя детьми с поддеревьями T_1, T_2 и T_3, то мы сперва приклеим v к поддереву T_1, потом приклеим компоненту из v и T_1 к поддереву T_2, потом приклеим получившуюся компоненту к T_3 и, тем самым, получим всё поддерево вершины v. При этом одна из склеиваемых компонент всегда была поддеревом какой-то вершины в выбранной ориентации дерева, так что наше многострадальное условие всегда выполняется.

Почему такой порядок склейки естественен? Для упрощения кода в динамиках по поддеревьям, внутри которых нужна «рюкзакоподобная» динамика по детям одной вершины, часто применяется такой же трюк: вместо использования «рюкзака» поддеревья детей приклеиваются по одному ко всё увеличивающемуся дереву.

Чтобы посчитать каждое дерево ровно один раз, достаточно зафиксировать для каждой вершины, в каком порядке мы приклеиваем к ней поддеревья детей. Мы не хотим, чтобы этот порядок сильно зависел от конкретного дерева, которое мы хотим в итоге получить (так как мы хотим делать эту динамику «одновременно по всем возможным итоговым деревьям»), поэтому сортировать по номерам детей — плохая идея. Гораздо лучше сортировать по чему-то, что зависит только от поддерева, а не от его корня, скажем, по убыванию наименьшего номера вершины в поддереве ребёнка (есть и другие «хорошие» порядки, но давайте остановимся на этом; «убывание» вместо «возрастания» чуть упростит будущие рассуждения).

У нас почти получилось решение! Давайте называть деревья, которые получаются в процессе склейки поддеревьев детей вершины v с ней, *частичными поддеревьями* v. Сама вершина v и всё поддерево v — тоже частичные поддеревья. На каждом шаге нашего процесса мы склеиваем какое-то частичное поддерево с каким-то настоящим поддеревом, при этом корень настоящего поддерева становится ребёнком корня частичного поддерева. При этом с точки зрения самих значений динамики нет никакого разделения между настоящими и частичными поддеревьями: зафиксировав порядок всех склеек, мы уже добились того, что для каждого итогового дерева есть ровно один порядок склейки компонент, который к нему приводит. Осталось лишь несколько мелких деталей, касающихся нумерации вершин.

Первая деталь, касающаяся нумерации: номера вершин внутри компоненты размера k могут быть произвольными различными числами от 1 до n, а хочется, чтобы они были от 1 до k. Чтобы разобраться с этим, «сожмём координаты», то есть будем считать количество компонент с фиксированным порядком вершин относительно их нумерации, не фиксируя нумерацию: тогда можно считать, что номера вершин лежат в отрезке $[1, k]$.

Вторая деталь: что происходит, когда мы подклеиваем настоящее поддерево размера full к частичному поддереву размера part? Вершина получившегося дерева с наименьшим номером среди всех вершин, кроме корня, будет лежать в настоящем поддереве, поскольку мы выбрали такой порядок подклейки детей (здесь случилось обещанное «чуть упростит»). Более того, когда мы строим итоговое дерево, мы всегда **знаем** корень текущего частичного, поэтому его номер тоже как-то зафиксирован. Наша свобода состоит в том, чтобы понять, как оставшиеся $\text{full} - 1$ и $\text{part} - 1$ номеров в настоящем и частичном поддеревьях соответственно соотносятся друг с другом ($C^{\text{full}-1}_{\text{full}+\text{part}-2}$ способов), а также в выборе того, за какую вершину мы подвешиваем настоящее поддерево к корню частичного: её уже нужно выбирать, так как разные вершины соответствуют разным схемам склейки, а соответственно, и разным итоговым деревьям (full способов). Итого,

$$\text{dp}(k, \text{sum}) = \sum_{\substack{\text{full} + \text{part} = k,\\ \text{fullsum} + \text{partsum} + \text{full} \cdot (n - \text{full}) = \text{sum}}} \text{dp}(\text{part}, \text{partsum}) \cdot \text{dp}(\text{full}, \text{fullsum}) \cdot C^{\text{full}-1}_{\text{full}+\text{part}-2} \cdot \text{full}$$

Первый параметр динамики — количество вершин в текущей компоненте, второй — сумма вкладов вершин из компоненты в итоговую сумму расстояний (напомню, что это не то же самое, что сумма расстояний между вершинами внутри компоненты, вклад также частично учитывает расстояния до вершин вне компоненты; именно из-за этого в решении возникало столько тонких моментов). Ответ лежит в $\text{dp}(n, 1), \text{dp}(n, 2), \ldots, \text{dp}(n, m)$. База динамики — деревья из одной вершины, из которых мы собираем итоговое дерево, то есть $\text{dp}(1, 0) = 1$, а остальные значения изначально равны 0. Конечно же, все операции надо делать по указанному в условии задачи модулю.

У этого решения есть небольшой недостаток. Поскольку второй параметр динамики имеет порядок n^3, самая простая его реализация работает за $O(n^8)$. Однако, оказывается что у этого решения очень маленькая константа. Например, значение второго параметра динамики всегда не больше $m \leqslant \frac{n^3}{6}$. То есть пар состояний динамики на самом деле не n^8, а не больше $\frac{n^8}{36}$. И это не единственное такое явление, на самом деле ненулевых значений динамики около $2 \cdot 10^5$, и так далее.

В зависимости от реализации это решение проходит все тесты либо сразу, либо после некоторого количества неасимптотических оптимизаций, либо после не очень долгого предподсчёта.

Несколько замечаний для подготовленных читателей:

1. Эту задачу можно решить за $O(n^5 \log n)$, $O(n^5)$ и даже $O(n^4 \log n)$. Для того, чтобы получить первую асимптотику, нужно заметить в формулах пересчёта умножение многочленов степени $O(n^3)$. Вторая и третья асимптотики требуют более хитрого подхода, и я рекомендую попытаться их достичь, если вам понравилась эта задача и вы уверены в своём понимании преобразования Фурье и смежных алгоритмов.

2. Можно ли решить задачу ещё быстрее? Скажем, за $O(n^4)$? Я не знаю ответа на этот вопрос.

3. Делая потестовую оценку, я надеялся, что будет много посылок с экспоненциальными решениями, работающими быстрее полного перебора всех помеченных деревьев. Например, есть такое решение, которое должно получать около 85 баллов: переберём все *непомеченные* деревья на n вершинах, для каждого из них посчитаем сумму расстояний и количество автоморфизмов. Это решение гораздо сложнее в реализации по сравнению с авторским, да и работает медленнее, но зато намного проще с идейной точки зрения.

4. Есть ли какая-то «явная математическая формула» для искомой величины? Нет почти никакой надежды на замкнутую формулу в привычном смысле этого слова, но существование какой-нибудь сложной формулы с определителями, суммами полиномиального количества слагаемых и тому подобными вещами ничему не противоречит.

5. Есть ли какой-то простой комбинаторный смысл (в отрыве от решаемой задачи) у промежуточных значений динамики или каких-то их простых комбинаций?

6. Я боялся, что эту задачу будет можно «загуглить», однако сам этого сделать не смог. Более того, все мои попытки найти ответы в OEIS не увенчались успехом, поэтому я всё-таки решил дать эту задачу. Если кто-нибудь справился «загуглить» эту задачу, то как вы это делали?